RDFS & OWL Reasoning for Linked Data

Axel Polleres[1], Aidan Hogan[2], Renaud Delbru[2], and Jürgen Umbrich[2,3]

[1] Siemens AG Österreich, Siemensstraße 90, 1210 Vienna, Austria
[2] Digital Enterprise Research Institute, National University of Ireland, Galway
[3] Fujitsu (Ireland) Limited, Swords, Co. Dublin, Ireland

Abstract. Linked Data promises that a large portion of Web Data will be usable as one big interlinked RDF database against which structured queries can be answered. In this lecture we will show how reasoning – using RDF Schema (RDFS) and the Web Ontology Language (OWL) – can help to obtain more complete answers for such queries over Linked Data. We first look at the extent to which RDFS and OWL features are being adopted on the Web. We then introduce two high-level architectures for query answering over Linked Data and outline how these can be enriched by (lightweight) RDFS and OWL reasoning, enumerating the main challenges faced and discussing reasoning methods that make practical and theoretical trade-offs to address these challenges. In the end, we also ask whether or not RDFS and OWL are enough and discuss numeric reasoning methods that are beyond the scope of these standards but that are often important when integrating Linked Data from several, heterogeneous sources.

1 Introduction

Linked Data [36, 2] denotes the emerging trend of publishing structured data on the Web as interlinked RDF graphs [49] following a number of simple principles: use HTTP URIs to name things, return RDF about those things when their URIs are looked up, and include links to related RDF documents elsewhere on the Web. These trends – and related trends with respect to embedding metadata into HTML pages (such as promoted by `schema.org`) – have lead to a huge volume of RDF data being made openly available online, contributing to what is often called the "Web of Data".

The Semantic Web provides the necessary tools to query this data, (i) firstly by defining RDF [49, 35] as a universal data format; (ii) secondly by defining SPARQL [60, 30], a standard query language for RDF; and (iii) lastly by providing schema languages such as RDF Schema (RDFS) [13] and OWL [37], which allow for adding rich semantics to the data.

It is the aim of this lecture to emphasise that all three components, and in particular the reasoning capabilities enabled by RDFS and OWL, are essential to enable usage of the Web of Data as "*one huge database*" as originally envisioned by Tim Berners-Lee [9].

As opposed to standard reasoning and query answering in OWL, for instance as described in the present volume in the chapter on Ontology-based Data Access [47], reasoning over Linked Data poses several unique challenges:

C1 *Linked Data is huge*, that is, it needs highly scalable or modular reasoning techniques;
C2 *Linked Data is not "pure" OWL*, that is, a lot of RDF Data published as Linked Data violates the strict syntactic corset of OWL (2) DL, and thus is not directly interpretable under OWL Direct Semantics;
C3 *Linked Data is inconsistent*, that is, if you take the Web of Data in its entirety, it is quite normal to encounter inconsistencies – not only from accidental or malicious datasets – but also because publishers may express contradicting views;
C4 *Linked Data is evolving*, that is, RDF Graphs on the Web evolve, they change, information is added and removed;
C5 *Linked Data needs more than RDFS and OWL*, that is, there is more implicit data hidden in Linked Data than can be captured with the semantics of RDFS and OWL alone.

In this lecture, we will introduce and discuss robust and scalable reasoning techniques that are specifically tailored to deal with these challenges in diverse and large-scale Linked Data settings. We first recapitulate the basic concepts of RDF, Linked Data, RDFS, OWL and SPARQL, and, with reference to a practical real-world scenario, we exemplify the use of query-rewriting techniques and rule-based approaches for reasoning over Linked Data (Section 2). We then will reflect in more detail on challenges C1–C5 above and discuss basic architectures, namely data-warehousing and on-the-fly-traversal based approaches, to reason about and query over Linked Data (Section 3).

While the W3C has defined two standard OWL fragments tailored for both rule-based (OWL 2 RL) and query-rewriting (OWL 2 QL) techniques, as we will see, standard reasoning within these fragments may not be a perfect fit for the Linked Data use-case; we will thus discuss which OWL features are actually predominantly used in current Linked Data and how these features relate to various standard and non-standard OWL fragments (Section 4).

The remaining chapters will then introduce specific approaches to Linked Data reasoning, each of which addresses some of the above-named challenges:

Section 5 introduces two rule-based approaches for Linked Data warehouses, both of which apply a cautious form of materialisation that considers where the axioms in question were found on the Web:
- *Context-Dependent Reasoning* [17], which is deployed within the Sindice semantic web engine [54].
- *Authoritative Reasoning* [41, 12], which is deployed within the Scalable Authoritative OWL Reasoner (SAOR) as part of the Semantic Web Search Engine (SWSE) [40].

Section 6 presents an alternative approach to enrich on-the-fly-traversal based approaches for Linked Data query processing with lightweight OWL Reasoning [67].

Section 7 presents a technique for reasoning with "attribute equations" [10] that models interdependencies between numerical properties expressible in terms of simple mathematical equations, which we argue is complementary

to RDFS/OWL techniques in that it allows for integrating numerical information embedded in Linked Data in a manner that RDFS and OWL do not.

We then briefly recap and conclude the lecture material in Section 8.

2 Preliminaries

As mentioned in the introduction, more and more Web sources provide readily accessible and interlinked RDF data. In the context of this paper, when speaking about structured data on the Web, we focus on data represented and published in the Resource Description Framework (RDF) according to the Linked Data principles. Hundreds of such datasets have been published in this manner, and have been collectively illustrated by Cyganiak and Jentzsch in the so-called "Linked Data Cloud".[4] Among these DBpedia[5] plays a central role as an RDF extract of structured data from Wikipedia. As a further example, the New York Times[6] provide RDF metadata about entities occurring in their articles as well as API access to the latest articles about these entities.

In terms of reasoning, our main considerations revolve around deductive inferences that enrich this RDF data with implicit information by means of exploiting (parts of) the formal semantics of RDFS and OWL. In particular, we aim at motivating how such additional inferences can contribute to query answering using SPARQL queries over the whole body of Web-accessible Linked Data (per Berners-Lee's "one big database" vision [9]).

We begin by introducing the basic concepts of the relevant Web standards along with a running example from real-world Linked Data. As a sort of disclaimer upfront, we emphasise that these preliminaries are not intended to replace a fully-fledged introduction to RDF, SPARQL, OWL or Description Logics; we refer the interested reader to excellent introductory chapters published within the earlier editions of the Reasoning Web summer school or to the official current W3C standard specifications in the references for further details.

2.1 The Resource Description Framework – RDF

Informally, all RDF data can be understood as a set of subject–predicate–object triples, where all subjects and predicates are URIs, and in the object position both URIs and literal values (such as numbers, strings, etc.) are allowed. Furthermore, blank nodes can be used in the subject or object resource to denote an unnamed resource with local scope. Some sample RDF data in the popular Turtle syntax [5, 6] are shown in Fig. 1.

[4] `http://lod-cloud.net`
[5] `http://dbpedia.org`
[6] `http://data.nytimes.org`

More formally, given the set of URI references U, the set of blank nodes B, and the set of literals L, the set of *RDF constants* is denoted by $\mathsf{C} := \mathsf{UBL}$.[7]. A triple $t := (s, p, o) \in \mathsf{UB} \times \mathsf{U} \times \mathsf{C}$ is called an *RDF triple*, where s is called subject, p predicate, and o object. A triple $t := (s, p, o) \in \mathsf{Tr}, \mathsf{Tr} := \mathsf{C} \times \mathsf{C} \times \mathsf{C}$ is called a *generalised triple* [25], which allows any RDF constant in any triple position: henceforth, we assume generalised triples unless explicitly stated otherwise. We call a finite set of triples $G \subset \mathsf{Tr}$ a *graph*.[8]

In the Turtle syntax, which we use for examples (see Fig. 1), URIs are often denoted either in full form using strings delimited by angle brackets (e.g. `<http://dbpedia.org/resource/Werner_von_Siemens>`) or as shorthand prefixed names (e.g. `dbr:Werner_von_Siemens`), where prefixes are as defined in Fig. 1. Literals are delimited by double quotes (e.g. `"SAP AG"`) with an optional trailing language tag (e.g. `@en`, `@de`) or a datatype which itself is again identified by a URI (e.g. `^^xsd:dateTime`). Blank nodes are (typically) scoped to a local document [35, 48] and are denoted in Turtle either by the "prefix" `_:` or alternatively using square brackets `[]`. Furthermore, Turtle allows use of '`a`' as a shortcut for `rdf:type` (denoting class membership) [5]. Finally, as shown in Fig. 1, RDF triples in Turtle notation are delimited by a trailing '`.`', where predicate-object-pairs that share the same subject can be grouped using '`;`' and several objects for the same subject–predicate pair can be grouped using '`,`'.

The data provided in Fig. 1 reflects real-world Linked Data available on the Web at the time of writing. However, we cheat slightly: in reality, revenue is not given in DBpedia using a dedicated property that includes the currency name (like `dbo:revenueUSD` or `dbo:revenueEUR`), but instead, e.g., the revenue data for IBM mentioned in Fig. 1(b) rather appears as follows:

```
dbr:IBM  dbp:revenue "US$ 106.916 billion";
         dbo:revenue "1.06916E11"^^dbdt:usDollar .
```

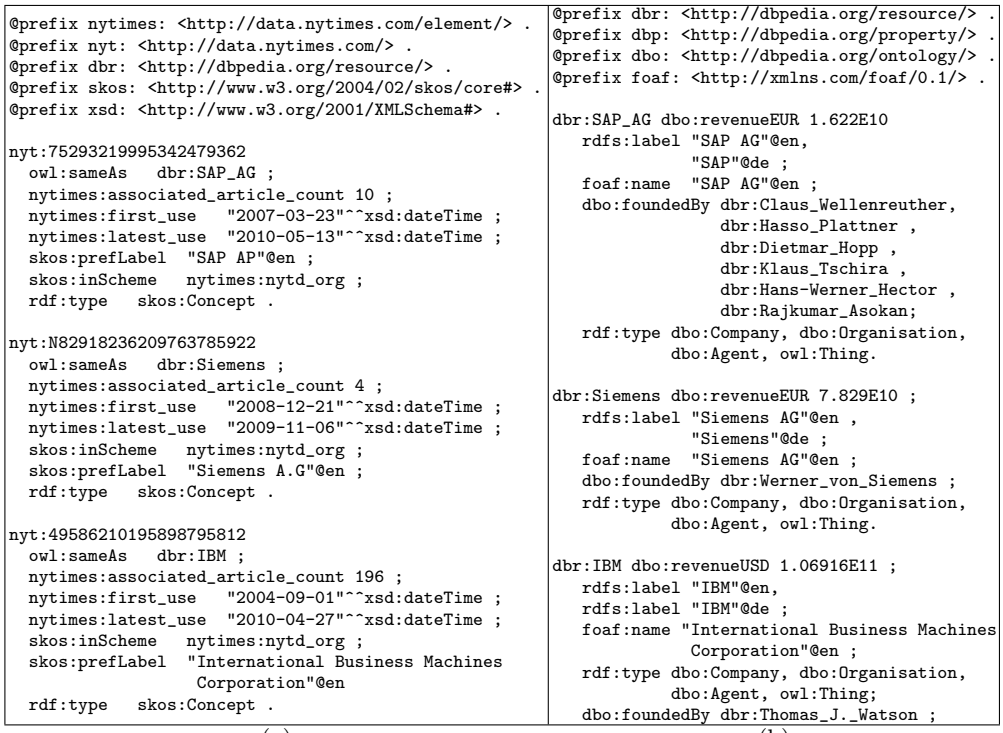
The use of different units (currencies in this case) and the fact that these units are often ill-defined for current Linked Data is a separate problem that we discuss later in Section 7.

2.2 Linked Data Principles and Provenance

In order to cope with the unique challenges of handling diverse and unverified RDF data spread over RDF graphs published at different URIs across the Web, many algorithms require inclusion of a notion of provenance, i.e., the consideration of the source of RDF data found on the Web. To this end, we provide some

[7] Herein, we sometimes use the convention of concatenating set names to represent unions; e.g. $\mathsf{UBL} = \mathsf{U} \cup \mathsf{B} \cup \mathsf{L}$. Also, though blank nodes are not constants by standard, we consider them as such for convenience. For more details on blank nodes and Skolemisation, we refer the interested reader to [48]

[8] Note that we use $\subset$ instead of $\subseteq$ here since we consider RDF graphs as *finite* subsets of the infinite set of triples Tr. We will also often use sans-serif (e.g., U) to denote an infinite set and use normal math font (e.g., U) to denote a finite proper subset.



```
@prefix nytimes: <http://data.nytimes.com/element/> .
@prefix nyt: <http://data.nytimes.com/> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

nyt:75293219995342479362
  owl:sameAs   dbr:SAP_AG ;
  nytimes:associated_article_count 10 ;
  nytimes:first_use   "2007-03-23"^^xsd:dateTime ;
  nytimes:latest_use  "2010-05-13"^^xsd:dateTime ;
  skos:prefLabel  "SAP AP"@en ;
  skos:inScheme   nytimes:nytd_org ;
  rdf:type   skos:Concept .

nyt:N82918236209763785922
  owl:sameAs   dbr:Siemens ;
  nytimes:associated_article_count 4 ;
  nytimes:first_use   "2008-12-21"^^xsd:dateTime ;
  nytimes:latest_use  "2009-11-06"^^xsd:dateTime ;
  skos:inScheme   nytimes:nytd_org ;
  skos:prefLabel  "Siemens A.G"@en ;
  rdf:type   skos:Concept .

nyt:49586210195898795812
  owl:sameAs   dbr:IBM ;
  nytimes:associated_article_count 196 ;
  nytimes:first_use   "2004-09-01"^^xsd:dateTime ;
  nytimes:latest_use  "2010-04-27"^^xsd:dateTime ;
  skos:inScheme   nytimes:nytd_org ;
  skos:prefLabel  "International Business Machines
                   Corporation"@en
  rdf:type   skos:Concept .
```

(a)

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

dbr:SAP_AG dbo:revenueEUR 1.622E10
   rdfs:label "SAP AG"@en,
              "SAP"@de ;
   foaf:name  "SAP AG"@en ;
   dbo:foundedBy dbr:Claus_Wellenreuther,
                 dbr:Hasso_Plattner ,
                 dbr:Dietmar_Hopp ,
                 dbr:Klaus_Tschira ,
                 dbr:Hans-Werner_Hector ,
                 dbr:Rajkumar_Asokan;
   rdf:type dbo:Company, dbo:Organisation,
            dbo:Agent, owl:Thing.

dbr:Siemens dbo:revenueEUR 7.829E10 ;
   rdfs:label "Siemens AG"@en ,
              "Siemens"@de ;
   foaf:name  "Siemens AG"@en ;
   dbo:foundedBy dbr:Werner_von_Siemens ;
   rdf:type dbo:Company, dbo:Organisation,
            dbo:Agent, owl:Thing.

dbr:IBM dbo:revenueUSD 1.06916E11 ;
   rdfs:label "IBM"@en,
   rdfs:label "IBM"@de ;
   foaf:name "International Business Machines
              Corporation"@en ;
   rdf:type dbo:Company, dbo:Organisation,
            dbo:Agent, owl:Thing;
   dbo:foundedBy dbr:Thomas_J._Watson ;
```

(b)

Fig. 1. Sample Data from DBpedia and NYT in Turtle Syntax (retrieved 10 March 2013)

formal preliminaries for the Linked Data principles, and HTTP mechanisms for retrieving RDF data from *dereferenceable URIs* (i.e., URIs that return content when looked up over HTTP).

Linked Data Principles Herein, we will refer to the four best practices of Linked Data publishing as follows [7]:

LDP1: use URIs as names for things;
LDP2: use HTTP URIs so those names can be dereferenced;
LDP3: return useful – herein we assume RDF – information upon dereferencing of those URIs; and
LDP4: include links using externally dereferenceable URIs.[9]

Data Source We define the *http-download* function $\mathsf{get} : \mathsf{U} \to 2^{\mathsf{Tr}}$ as the mapping from a URI to an RDF graph it provides by means of a given HTTP lookup [19] which directly returns status code `200 OK` and data in a suitable RDF format, or

[9] That is, within your published RDF graph, use HTTP URIs pointing to other dereferenceable documents, that possibly contain further RDF graphs.

to the empty set in the case of failure; this function also performs a rewriting of blank-node labels (based on the input URI) to ensure uniqueness when merging RDF graphs [35]. We define the set of *data sources* $\mathsf{S} \subset \mathsf{U}$ as the set of URIs $\mathsf{S} := \{s \in \mathsf{U} \mid \mathsf{get}(s) \neq \emptyset\}$.[10]

RDF Triple in Context/RDF Quadruple An ordered pair (t, c) with a triple $t := (s, p, o)$, and with a context $c \in \mathsf{S}$ and $t \in \mathsf{get}(c)$ is called a *triple in context* c. We may also refer to (s, p, o, c) as an *RDF quadruple* or quad q with context c. In Fig. 2 we illustrate the RDF data from Fig. 1 as dereferenceable RDF graphs. Note that Fig. 2 contains more data than Fig. 1, where we include triples obtained from dereferencing the founders of organisations mentioned in Fig. 1, as well as data obtained from dereferencing class and property URIs.

HTTP Dereferencing/Redirects We now give some notation relating to a "Linked Dataset", which is inspired by the notion of named graphs in SPARQL, but where the dataset reflects the $\mathsf{get}(\cdot)$ function that maps graph names to RDF graphs obtained from dereferencing URIs.

Definition 1 (Data Source and Linked Dataset).
We define a Linked Dataset as $\Gamma \subset \mathsf{get}$; i.e., a finite set of pairs $(s, \mathsf{get}(s))$ such that $s \in \mathsf{S}$.[11] *The "global" RDF graph presented by a Linked Dataset is denoted as*

$$\mathsf{merge}(\Gamma) := \biguplus_{(u,G)\in\Gamma} G$$

where the operator '$\uplus$' denotes the RDF merge of RDF graphs: a set union where blank nodes are rewritten to ensure that no two input graphs contain the same blank node label [35]. (From the perspective of a SPARQL dataset, one may consider $\mathsf{merge}(\Gamma)$ as the "default graph" and all such $(s, \mathsf{get}(s))$ as named graph pairs.)

A URI may provide a HTTP redirect to another URI using a `30x` response code [19]; we denote this function as $\mathsf{redir} : \mathsf{U} \to \mathsf{U}$ which first removes any fragment identifier from the URI (a hash string suffix) and then follows a single redirect, mapping a URI to itself in the case of failure (e.g., where no redirect exists). We denote the fixpoint of redir as redirs, denoting traversal of a number of redirects (a limit may be set on this traversal to avoid cycles and artificially long redirect paths). We define *dereferencing* as the function $\mathsf{deref} := \mathsf{get} \circ \mathsf{redirs}$ which maps a URI to an RDF graph retrieved with status code `200 OK` *after* following redirects, or which maps a URI to the empty set in the case of failure.

We denote the set of *dereferenceable URIs* as $\mathsf{D} := \{d \in \mathsf{U} : \mathsf{deref}(d) \neq \emptyset\}$; note that $\mathsf{S} \subset \mathsf{D}$ and we place no expectations on what $\mathsf{deref}(d)$ returns, other

[10] Note again that we use $\subset$ instead of $\subseteq$ to emphasise that obviously not all URIs point to non-empty RDF graphs.
[11] Γ is finite thus – again – we use $\subset$ instead of $\subseteq$.

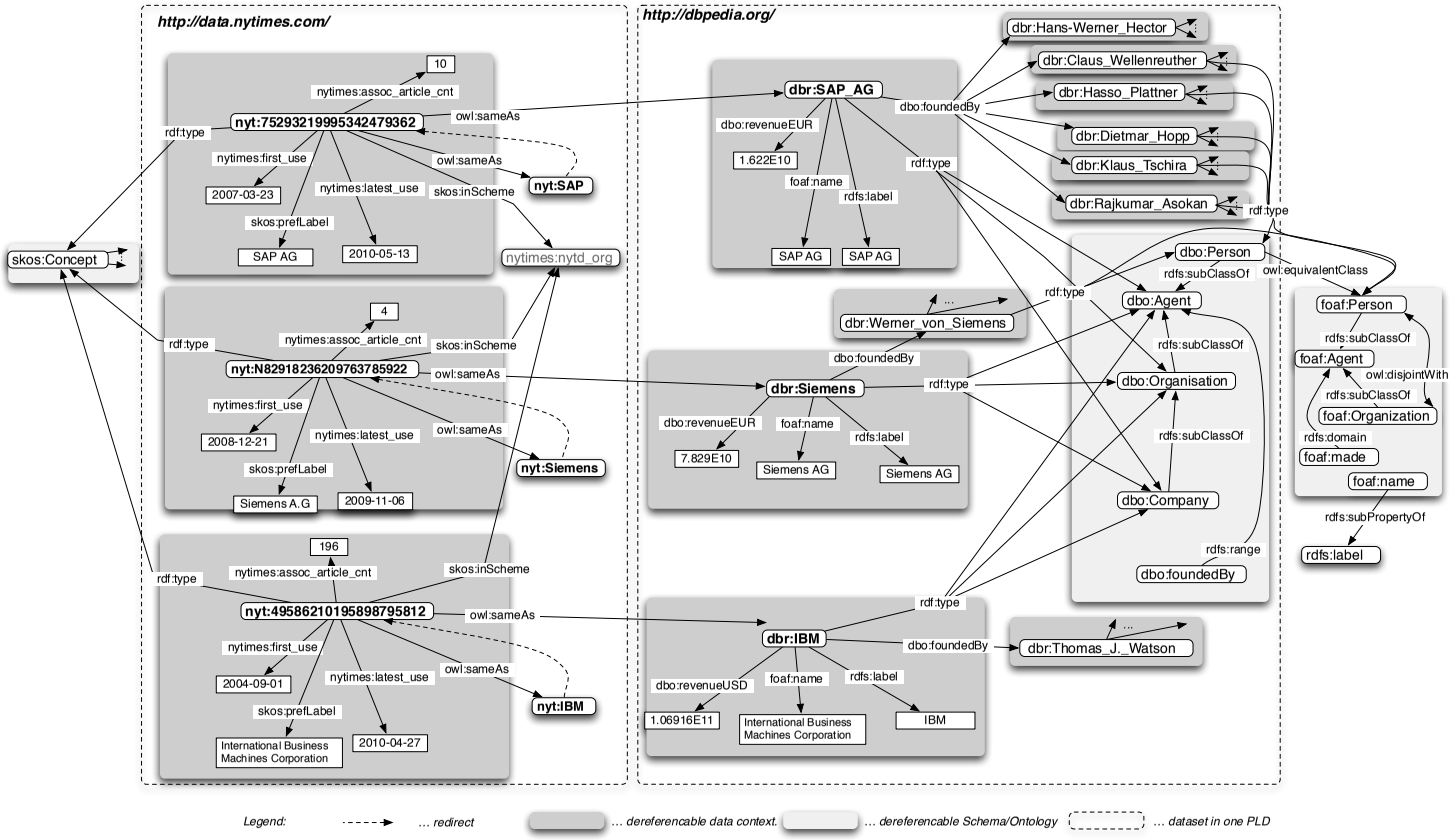


Fig. 2. Sample Data from DBpedia and NYT as Linked Data

than returning *some* valid RDF.[12] As a shortcut, we denote by $\mathsf{derefs} : 2^{\mathsf{U}} \to 2^{\mathsf{U} \times 2^{\mathsf{Tr}}}; U \mapsto \{(\mathsf{redirs}(u), \mathsf{deref}(u)) \mid u \in U \cap \mathsf{D})\}$ the mapping from a set of URIs to the Linked Dataset it represents by dereferencing all URIs (only including those in D which return some RDF).

Example 1

Taking Fig. 2, dashed arrows indicate redirects: NYT provides some "readable" URIs that redirect to its internal identifiers for entities, including $\mathsf{redirs}(\texttt{nyt:Siemens}) = \texttt{nyt:N82918236209763785922}$. Dereferencing the URI `nyt:Siemens` yields the RDF graph for the source `nyt:N82918236209763785922` (symbolised by the gray box in Fig. 2); in other words $\mathsf{deref}(\texttt{nyt:Siemens}) = \mathsf{get}\big(\mathsf{redirs}(\texttt{nyt:Siemens})\big) = \mathsf{get}(\texttt{nyt:N82918236209763785922})$. Conversely, we see $\mathsf{deref}(\texttt{nytimes:nytd\_org}) = \emptyset$; the URI is not dereferenceable. Thus we say that $\texttt{nyt:N82918236209763785922} \in \mathsf{D}$ (and $\in \mathsf{S}$) and $\texttt{nyt:Siemens} \in \mathsf{D}$, whereas $\texttt{nytimes:nytd\_org} \notin \mathsf{D}$. ◊

2.3 SPARQL, Conjunctive Queries and Rules

We now introduce some concepts relating to the query language SPARQL, where details can be found in [60, 57, 30]. We herein focus on evaluating simple, conjunctive, *basic graph patterns* (BGPs), we do not formally consider more complex patterns (e.g., `UNION`, `FILTER`, etc.) of the SPARQL language, which can be layered on top [57]. In addition, we consider URIs and not IRIs for convenience and to stay consistent with the RDF preliminaries.

Definition 2 (Variables, Triple Patterns & BGPs).
Let V *be the set of variables ranging over* UBL*; we denote RDF variables as alphanumeric strings with a '?' prefix (e.g.* `?X`*). A* triple pattern $tp := (s, p, o)$ *is an element of the set* $\mathsf{Q} := \mathsf{VUL} \times \mathsf{VU} \times \mathsf{VUL}$.[13] *For simplicity, we do not consider blank-nodes in triple patterns (they could be roughly emulated by an injective mapping from* B *to* V*). A finite (herein, non-empty) set of triple patterns* $Q \subset \mathsf{Q}$ *is called a* Basic Graph Pattern *(BGP), or herein, simply a* conjunctive query*. We use* $\mathsf{vars}(Q) \subset \mathsf{V}$ *to denote the finite set of variables in* Q *and* $\mathsf{terms}(Q)$ *to denote the set of all terms* VUL *in* Q*.*

The solutions of a BGP for a dataset Γ are defined accordingly as follows.

Definition 3 (SPARQL solutions).
Call the partial function $\mu : \mathrm{dom}(\mu) \cup \mathsf{UL} \to \mathsf{UBL}$ *a* solution mapping *with a*

[12] The URIs within D that are not in S are those that return (non-empty) RDF graphs by means of (an) intermediate redirect(s) and/or after having a fragment identifier stripped.

[13] Though academic, SPARQL does allow for literals in the subject position.

domain $\mathrm{dom}(\mu) \subseteq \mathsf{V}$. *A solution mapping binds variables in* $\mathrm{dom}(\mu)$ *to* UBL *and is the identify function for* UL. *Overloading notation, let* $\mu : \mathsf{Q} \to \mathsf{Tr}$ *and* $\mu : 2^{\mathsf{Q}} \to 2^{\mathsf{Tr}}$ *also resp. denote a solution mapping from triple patterns to RDF triples, and basic graph patterns to RDF graphs such that* $\mu(tp) \coloneqq (\mu(s), \mu(p), \mu(o))$ *and* $\mu(Q) \coloneqq \{\mu(tp) \mid tp \in Q\}$. *We now define the set of* SPARQL solutions *for a query* Q *over a (Linked) Dataset* Γ *as*

$$[\![Q]\!]_\Gamma \coloneqq \{\mu \mid \mu(Q) \subseteq \mathsf{merge}(\Gamma) \wedge \mathrm{dom}(\mu) = \mathsf{vars}(Q)\}\,.$$

For brevity, and unlike SPARQL as according to the official W3C specification, solutions are herein given as sets (not multi-sets), implying a default `DISTINCT` *semantics for queries, and we assume that answers are given over the default graph consisting of the merge of RDF graphs in the dataset.*

Note that in the database literature, conjunctive queries are often rather modelled as conjunctions of (first-order) atoms, inspired by languages such as Datalog; let us introduce some basic concepts from Datalog to relate the notions of BGPs, conjunctive queries, and rules:

Definition 4 (Atom).
An atomic formula or atom is a formula of the form $p(e_1, \ldots, e_n)$*, where all such* $e_1, \ldots, e_n$ *are terms (i.e. in our case constants from* C *or variables from* V*) and where* p *is a predicate of arity* n *– we denote the set of all such atoms by* Atoms*. Atoms that to not contain variables are called ground. As such, this notation can be thought of as generalising that of RDF triples (or quadruples, resp.), where we use a standard RDF* ternary predicate T *to represent RDF triples in the form* $T(s, p, o)$ *– for example,* T`(Siemens, foundedby, Werner_von_Siemens)` *– where we will typically leave* T *implicit; analogously, we use the predicate* $Q(s, p, o, c)$ *synonymously for an RDF quadruple, again optionally leaving* Q *implicit.*

In Logic Programming, atoms not containing variables are called *ground atoms*. The set of all possible ground atoms – denoted as the set Facts – can be viewed in our context as a generalisation of Tr. A finite set of ground atoms $I \subseteq \mathsf{Facts}$ (often simply called "*facts*") – which in our context again can be viewed as a generalisation of a graph G – in Logic Programming is typically synonymous to a (Herbrand) *interpretation*.

Letting A and B be two atoms, we say that A *subsumes* B—denoted $A \triangleright B$—if there exists a substitution $\theta \in \Theta$ of variables such that $A\theta = B$ (applying θ to the variables of A yields B); we may also say that B is an *instance* of A; if B is ground, we say that it is a *ground instance*. Similarly, if we have a substitution $\theta \in \Theta$ such that $A\theta = B\theta$, we say that θ is a *unifier* of A and B; we denote by $\mathsf{mgu}(A, B)$ the *most general unifier* of A and B which provides the "minimal" variable substitution (up to variable renaming) required to unify A and B.

Definition 5 (Rule, Program). *A* rule R *is given as follows:*

$$H \leftarrow B_1, \ldots, B_n (n \geq 0)\,,$$

where $H, B_1, \ldots, B_n$ *are atoms,* H *is called the* head *(conclusion/consequent) and* $B_1, \ldots, B_n$ *the* body *(premise/antecedent). We use* $\mathsf{Head}(R)$ *to denote the head* H *of* R *and* $\mathsf{Body}(R)$ *to denote the body* $B_1, \ldots, B_n$ *of* R.[14] *The variables of our rules are range restricted, also known as safe [66]: like Datalog, the variables appearing in the head of each rule must also appear in the body whereby a substitution that grounds the body must also ground the head. We denote the set of all such rules by* Rules. *A rule with an empty body can be considered a fact; a rule with a non-empty body is called a proper-rule. We call a finite set of such rules a* program P.

Like before, a ground rule is one without variables. We denote with $\mathsf{Ground}(R)$ the set of ground instantiations of a rule R and with $\mathsf{Ground}(P)$ the ground instantiations of all rules occurring in a program P. Again, an *RDF rule* is a specialisation of the above rule, where atoms strictly have the ternary predicate T and contain RDF terms; an *RDF program* is one containing RDF rules, etc.

Definition 6 (Immediate Consequence Operator). *We give the immediate consequence operator* $\mathfrak{T}_P$ *of a program* P *under interpretation* I *as:*[15]

$$\mathfrak{T}_P : \ 2^{\mathsf{Facts}} \to 2^{\mathsf{Facts}}$$
$$I \mapsto \left\{ \mathsf{Head}(R)\theta \mid R \in P \wedge \exists I' \subseteq I \text{ s.t. } \theta = \mathsf{mgu}\big(\mathsf{Body}(R), I'\big) \right\}$$

The immediate consequence operator maps from a set of facts I to the set of facts it directly entails with respect to the program P – note that $\mathfrak{T}_P(I)$ will retain the facts in P since facts are rules with empty bodies and thus unify with any interpretation, and note that $\mathfrak{T}_P$ is *monotonic* – the addition of facts and rules to a program can only lead to the same or additional consequences. We may refer to the application of a single rule $\mathfrak{T}_{\{R\}}$ as a *rule application.*

Since our rules are a syntactic subset of Datalog, $\mathfrak{T}_P$ has a *least fixpoint*, denoted $\mathsf{lfp}(\mathfrak{T}_P)$, which can be calculated in a bottom-up fashion, starting from the empty interpretation Δ and applying iteratively $\mathfrak{T}_P$ [73] (here, convention assumes that P contains the set of input facts as well as proper rules). Define the iterations of $\mathfrak{T}_P$ as follows: $\mathfrak{T}_P \uparrow 0 = \Delta$; for all ordinals α, $\mathfrak{T}_P \uparrow (\alpha + 1) = \mathfrak{T}_P(\mathfrak{T}_P \uparrow \alpha)$; since our rules are Datalog, there exists an α such that $\mathsf{lfp}(\mathfrak{T}_P) = \mathfrak{T}_P \uparrow \alpha$ for $\alpha < \omega$, where ω denotes the least infinite ordinal. In other words, the immediate consequence operator will reach a fixpoint in countable steps [66]. Thus, $\mathfrak{T}_P$ is also *continuous.*

Definition 7 (least model, closure). *We call* $\mathsf{lfp}(\mathfrak{T}_P)$ *the least model, or the closure of* P*, which is given the more succinct notation* $\mathsf{lm}(P)$.

[14] Such a rule can be represented as a definite Horn clause.

[15] Recall again that in Herbrand semantics, an interpretation I can be thought of as simply a set of facts.

Obviously, using the notation of rules, any BGP can be straightforwardly seen as a conjunctive query (CQ) in the classical sense, and the more general concept of a union of conjunctive queries (UCQs) can be defined as follows.

Definition 8 (Union of Conjunctive Queries (classical notion)). *A* conjunctive query *(CQ) is a special rule of the form*

$$q(\vec{x}) \leftarrow Body(\vec{x}, \vec{y})$$

where $\vec{x}$ is a sequence of variables called distinguished variables*, $\vec{y}$ is a sequence of variables called* non-distinguished variables*, and $Body(\vec{x}, \vec{y})$ is a conjunction of body atoms over these variables. A program P_q (where we just write q when P is implicit form the context) that consists of only rules with the same head $q(\vec{x})$ (such that q does not appear in any body) is a* union of conjunctive queries *(UCQ).*

Given a (Linked) Dataset Γ and a SPARQL BGP query Q, the definition of solutions from Definition 3 thus corresponds to the notion of entailed answers for the corresponding classical conjunctive query q, written $\mathsf{Ans}(q, \mathsf{merge}(\Gamma))$ (and used in the literature, cf. for instance [14]).[16] That is, in our context we can equate $\mathsf{Ans}(q, \mathsf{merge}(\Gamma))$ with the set of tuples $\vec{a}$ such that $q(\vec{a}) \in \mathfrak{T}_{P_q}(\mathsf{merge}(\Gamma))$. Note that as for SPARQL, all variables are considered to be distinguished, since there is no "real" projection in the sense of classical conjunctive queries: any BGP occurring inside a SPARQL query is evaluated as a conjunctive query without non-distinguished variables, whereupon the SPARQL algebra evaluates more complex patterns, such as `SELECT` clauses and so forth [30, 21].

Example 2

The following SPARQL query asks for the labels and revenues (in EUR) of organisations:

QUERY 1

```
SELECT ?X ?L ?R
WHERE { ?X a dbo:Organisation ; rdfs:label ?L ; dbo:revenueEUR ?R .}
```

This query asks to evaluating the basic graph pattern:

$$\{(?X, \mathsf{a}, \mathsf{dbo{:}Organisation}), (?X, \mathsf{rdfs{:}label}, ?L), (?X, \mathsf{dbo{:}revenueEUR}, ?R)\}$$

which, respectively, in the classical notation corresponds to the following conjunctive query:

$$q(?X, ?L, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Organisation}), (?X, \mathsf{rdfs{:}label}, ?L), (?X, \mathsf{dbo{:}revenueEUR}, ?R).$$

[16] We use $\mathsf{merge}(\Gamma)$ here synonymously with the knowledge base consisting of the facts corresponding to the triples in $\mathsf{merge}(\Gamma)$.

Given the data in Fig. 2, this pattern (and thus the query) would obtain the following solutions (writing solution mappings μ in tabular form):

?X	?L	?R
dbr:SAP_AG	"SAP AG"@en	1.622E10
dbr:Siemens	"Siemens"@de	7.829E10

Notably, the revenue for IBM is not returned (although it could be calculated from the EUR–USD exchange rate).

The next query, on the contrary, asks for the date of the latest article published about each element in the SKOS scheme `nytimes:nytd_org` (i.e., a different way of asking for "organisations", but using the NYT RDF vocabulary):

QUERY 2
```
SELECT ?X ?D
WHERE { ?X skos:inScheme nytimes:nytd_org. ?X nytimes:latest_use ?D .}
```

with the following solutions:

?X	?D
nyt:75293219995342479362	2010-05-13
nyt:N82918236209763785922	2009-11-06
nyt:49586210195898795812	2010-04-27

Query 1 and Query 2 could each be answered by staying within a single site (that is, Query 1 only would obtain answers from data at `dbpedia.org`, whereas Query 2 would only produce answers with the data at `data.nytimes.org`, respectively) and – at least for our sample – answers for either query can be obtained from the individual dataset. However, the real power of Linked Data lies in combining data from multiple datasets to obtain answers over their combined content. For example Query 3 combines knowledge from both sites and asks for the latest NYT article dates of IBM (using its NYT identifier) and its revenue in USD:

QUERY 3
```
SELECT ?X ?D ?R
WHERE { nyt:49586210195898795812 nytimes:latest_use ?D .
        nyt:49586210195898795812 owl:sameAs ?X .
        ?X dbo:revenueUSD ?R .}
```

Again assuming the entire graph of Fig. 2 as input, this query would obtain the single result

?X	?D	?R
dbr:IBM	2010-04-27	1.06916E11

As a further example, let Query 4 ask for all `foaf:Agent`s.

QUERY 4
```
SELECT ?X
WHERE { ?X a foaf:Agent .}
```

Clearly, for the RDF data in Fig. 2, this query would not return any solutions (despite many implicit instances of the class being available), whereas:

QUERY 5
```
SELECT ?X
WHERE { ?X a foaf:Person .}
```

would return all the company founders listed in DBpedia, since these are explicitly typed as `foaf:Person`s. ◊

We emphasise that these queries miss (implicit) results that would be intuitively expected as an answer and that could be achieved through reasoning. In the following we will substantiate this intuition referring to some of the challenges (C1–C5) mentioned in the introduction; before that, however, we need to clarify the importance of schema information and its semantics in RDF.

2.4 Inferring Additional Triples by Schema Information and Rules

In order to model schema information, which also allows to infer additional implicit information in RDF, the Semantic Web offers two ontology languages: the lightweight RDF Schema (RDFS) [13] and the expressive Web Ontology Language (OWL) [37]. Within this section, we will briefly cover an overview of the essential features of these languages in a Linked Data setting.

RDFS RDF itself already provides means to express class membership (`rdf:type`); RDF Schema (RDFS) additionally provides a special vocabulary, consisting primarily of RDF properties with a predefined semantics to model class hierarchies (`rdfs:subClassOf`), and property hierarchies (`rdfs:subPropertyOf`), as well as means to define domains and ranges that respectively allow for associating a class to the subject and object of a relation with a given property (`rdfs:domain`, `rdfs:range`). These core RDFS properties allow for describing and embedding the semantics of user-defined vocabularies in RDF itself.

OWL The Web Ontology Language (OWL) extends RDFS and allows for expressing further schema definitions in RDF, e.g., allowing to express equality of individuals (`owl:sameAs`), equivalence or disjointness of properties and classes (`owl:equivalentClass`, `owl:equivalentProperty`, `owl:disjointWith`, `owl:propertyDisjointWith`), or complex class definitions; while a full account is beyond our scope, more details on additional OWL features will be discussed in Section 4 below.

Some RDFS and OWL information expressed in RDF is shown in the light-gray boxes of Fig. 2: for instance, the classes and properties used on DBpedia

are described using, amongst others, the `dbo:` and `foaf:` vocabularies (i.e., sets of terms described in their respective ontologies) which include the "schema" triples (aka. terminological triples) shown in Fig. 3.

```
@prefix dbo: <http://dbpedia.org/ontology/> .

dbo:foundedBy rdfs:range dbo:Agent .

dbo:Company rdfs:subClassOf dbo:Organisation.
dbo:Organisation rdfs:subClassOf dbo:Agent.
dbo:Person rdfs:subClassOf dbo:Agent.

dbo:Person owl:equivalentClass foaf:Person.
```

(a)

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

foaf:name rdfs:subPropertyOf rdfs:label .
foaf:made rdfs:domain foaf:Agent .

foaf:Person rdfs:subClassOf foaf:Agent ,
      geo:SpatialThing ;
   owl:disjointWith foaf:Organization .
foaf:Organization rdfs:subClassOf foaf:Agent .
```

(b)

Fig. 3. Sample schema triples from the DBpedia ontology and from FOAF (retrieved 10 March 2013)

The meaning of these OWL definitions are given by two alternative (but related) semantics. The RDF-Based Semantics [64] can interpret arbitrary RDF graphs without restriction, but where common reasoning tasks are undecidable as a result. The Direct Semantics [52] is based on Description Logic (DL) theory [3, 63], where a subset of RDF graphs following certain syntactic restrictions can be mapped to a set of DL axioms. A full translation from the OWL 2 DL language to the DL $\mathcal{SROIQ}$ is implicit in [56], where we list a small snippet of constructs used for our examples in Table 1. As such, according to Table 1, the RDF triples in Fig. 3 can be viewed as DL ontologies containing the axioms listed in Fig. 4.

Typical reasoning tasks over an expressive DL like $\mathcal{SROIQ}$ (e.g., using tableau methods to perform consistency checking, instance checking, satisfiability checking, etc.; see [3, 63]) are in the worst case doubly-exponential even for a non-deterministic machine, and in practice are often likewise very expensive, especially at the types of scales encountered in a Linked Data setting. Furthermore, the decidability of conjunctive query answering for $\mathcal{SROIQ}$ is still

Table 1. Mapping DL axioms to RDF using the RDFS and OWL vocabularies

	DL	RDFS
1	$A_1 \sqsubseteq A_2$	A_1 `rdfs:subClassOf` A_2
2	$P_1 \sqsubseteq P_2$	P_1 `rdfs:subPropertyOf` P_2
3	$\exists P \sqsubseteq A$	P `rdfs:domain` A
4	$\exists P^- \sqsubseteq A$	P `rdfs:range` A
5	$A_1 \equiv A_2$	A_1 `owl:equivalentClass` A_2
6	$A_1 \sqcap A_2 \sqsubseteq \bot$	A_1 `owl:disjointWith` A_2
7	$A(x)$	x rdf:type A
8	$R(x, y)$	x R y
9	$x = y$	x owl:sameAs y

$\exists$dbo:fundedby$^-$ $\sqsubseteq$ dbo:Agent	foaf:name $\sqsubseteq$ rdfs:label
dbo:Company $\sqsubseteq$ dbo:Organisation	$\exists$foaf:made $\sqsubseteq$ foaf:Agent
dbo:Organisation $\sqsubseteq$ dbo:Agent	foaf:Person $\sqsubseteq$ foaf:Agent
dbo:Person $\sqsubseteq$ dbo:Agent	foaf:Person $\sqsubseteq$ geo:SpatialThing
dbo:Person $\equiv$ foaf:Person	foaf:Person $\sqcap$ foaf:Organisation $\sqsubseteq \bot$
	foaf:Organization $\sqsubseteq$ foaf:Agent
(a)	(b)

Fig. 4. DL axioms corresponding to the DBpedia and FOAF ontology snippets from Fig. 3

open [25].[17] Thus, the W3C has identified three tractable profiles of OWL that are particularly relevant for query answering [25], where we will focus upon two such profiles in this lecture:

OWL 2 QL is designed as a language for which efficient (with respect to data complexity), sound and complete query answering can be achieved by rewriting techniques – i.e., extending conjunctive queries (such as SPARQL BGPs) to capture the semantics of the ontology.

OWL 2 RL is designed as a language for which sound and complete (with respect to assertional knowledge) reasoning can be applied by means of Datalog-style techniques – e.g., bottom-up or top-down rule-based inferencing. A standard OWL 2 RL ruleset, called OWL 2 RL/RDF, encodes part of the RDF-Based Semantics of OWL 2. The assertional entailments given by rule-based reasoning using OWL 2 RL/RDF over an OWL 2 RL ontology correspond with the assertional entailments under the Direct Semantics for that ontology [25, Theorem PR1]. The OWL 2 RL/RDF ruleset can also be applied over arbitrary RDF graphs but beyond OWL 2 RL ontologies, the aforementioned correspondence with the Direct Semantics no longer holds.

Query Rewriting The OWL 2 QL fragment [25, Section 3] contains a combination of features that are tailored for efficient processing by query rewriting techniques. That is, given an OWL (or, respectively, its DL counterpart) ontology O in this fragment, one can answer conjunctive queries Q correctly by rewriting Q based on the axioms in O into a union of conjunctive queries (a UCQ), where we denote this process by $rewrite(Q, O)$.

To give the reader an idea, Algorithm 1 illustrates a very rudimentary version of a rewriting algorithm that implements $rewrite(Q, O)$ just for the basic RDFS axioms (axioms of the forms 1–4 from Table 1). We note that this algorithm can be viewed as a very downstripped version of the PerfectRef algorithm [14] which covers a much larger set of DL axioms; there have been a number of extensions and alternative query rewriting techniques proposed recently [58, 62, 61, 46, 24].

[17] In this context, we note here that the case of SPARQL is decidable, since SPARQL's BGP matching treats all variables as distinguished, see above; for further details, we refer to the SPARQL 1.1 Entailment Regimes specification [23] and a more detailed book chapter in an earlier edition of the Reasoning Web summer school [21].

Algorithm 1: Basic Query Rewriting algorithm

Input: Conjunctive query q, DL ontology $\mathcal{O}$
Output: Union (set) of conjunctive queries

$P := P_q$
repeat
  $P' := P$
  foreach $q \in P'$ **do**
    foreach *g in q* **do** // expansion
      foreach *axiom i of one of the forms 1–4 from Table 1 in* $\mathcal{O}$ **do**
        if *i is applicable to g* **then**
          $P := P \cup \{q[g/\,\mathrm{gr}(g,i)]\}$ // see Table 2
until $P' = P$
return P

Table 2. Semantics of $\mathrm{gr}(g,i)$ of Algorithm 1 ('_' stands for a "fresh" variable)

g	i	$\mathrm{gr}(g/i)$
$(x, \mathsf{rdf{:}type}, A)$	$B \sqsubseteq A$	$(x, \mathsf{rdf{:}type}, B)$
	$\exists P \sqsubseteq A$	$P(x, \_)$
	$\exists P^- \sqsubseteq A$	$P(\_, x)$
(x, P_1, y)	$P_2 \sqsubseteq P_1$	(x, P_2, y)

Example 3

Taking Query 4 again, the BGP $\{(?X, \mathsf{a}, \mathsf{foaf{:}Agent})\}$ corresponds to the conjunctive query:

$$q(?X) \leftarrow (?X, \mathsf{rdf{:}type}, \mathsf{foaf{:}Agent})$$

which expanded by Algorithm 1 with respect to the ontology in Fig. 4(b) results in the following UCQ:

$$\begin{aligned} q(?X) &\leftarrow (?X, \mathsf{a}, \mathsf{foaf{:}Agent}) \\ q(?X) &\leftarrow (?X, \mathsf{a}, \mathsf{foaf{:}Person}) \\ q(?X) &\leftarrow (?X, \mathsf{a}, \mathsf{foaf{:}Organization}) \\ q(?X) &\leftarrow (?X, \mathsf{foaf{:}made}, ?Y) \end{aligned}$$

The resulting UCQ can again (using the UNION pattern) be translated back to SPARQL:

QUERY 4′

```
SELECT ?X
WHERE { { ?X a foaf:Agent } UNION
        { ?X a foaf:Person } UNION
        { ?X a foaf:Organization } UNION
        { ?X foaf:made ?Y } }
```

◊

With this small example, we have shown that the rewriting technique for OWL 2 QL can be partially applied to SPARQL queries. However, note that OWL 2 QL (and likewise the respective query rewriting algorithms from the DL literature) omit important OWL features for Linked Data (discussed later in Section 4), such as inferences from `owl:sameAs`, where rule-based inference as mentioned in the following section might be more suitable.

Rule-based Reasoning As an alternative to query rewriting based on the OWL 2 QL profile, another fragment of OWL – OWL 2 RL [25, Section 4] – has a normative set of rules called OWL 2 RL/RDF, which encode a subset of the semantics of RDFS and OWL 2 and can be directly used for (Datalog-style) reasoning (see also the informal RDFS entailment rules in [35, Section 7], which were later extended by OWL features in [65, 53]).

Some sample OWL 2 RL/RDF rules are given in Table 3 implementing the basic features of RDFS and additionally supporting the semantics of equality for `owl:sameAs` missed by OWL 2 QL. A more detailed discussion on which OWL features (and respective rules encoding these) are particularly relevant for Linked Data Reasoning will be discussed in Section 4.

Table 3. Core RDFS and `owl:sameAs` rules

	ID	Head	Body
RDFS	prp-spo1	$(?s, ?p_2, ?o)$	$\leftarrow (?p_1, \mathsf{rdfs{:}subPropertyOf}, ?p_2), (?s, ?p_1, ?o)$
	prp-dom	$(?p, \mathsf{rdf{:}type}, ?c)$	$\leftarrow (?p, \mathsf{rdfs{:}domain}, ?c), (?s, ?p, ?o)$
	prp-rng	$(?o, \mathsf{rdf{:}type}, ?c)$	$\leftarrow (?p, \mathsf{rdfs{:}range}, ?c), (?s, ?p, ?o)$
	cax-sco	$(?s, \mathsf{rdf{:}type}, ?c_2)$	$\leftarrow (?c_1, \mathsf{rdfs{:}subClassOf}, ?c_2), (?s, \mathsf{rdf{:}type}, ?c_1)$
Same-As	eq-sym	$(?y, \mathsf{owl{:}sameAs}, ?x)$	$\leftarrow (?x, \mathsf{owl{:}sameAs}, ?y)$
	eq-trans	$(?x, \mathsf{owl{:}sameAs}, ?z)$	$\leftarrow (?x, \mathsf{owl{:}sameAs}, ?y), (?y, \mathsf{owl{:}sameAs}, ?z)$
	eq-rep-s	$(?s', ?p, ?o)$	$\leftarrow (?s, \mathsf{owl{:}sameAs}, ?s'), (?s, ?p, ?o)$
	eq-rep-p	$(?s, ?p', ?o)$	$\leftarrow (?p, \mathsf{owl{:}sameAs}, ?p'), (?s, ?p, ?o)$
	eq-rep-o	$(?s, ?p, ?o')$	$\leftarrow (?o, \mathsf{owl{:}sameAs}, ?o'), (?s, ?p, ?o)$

As opposed to query-rewriting (top-down evaluation), OWL 2 RL/RDF rules can also be applied in a bottom-up fashion for the purposes of a priori *materialisation*: given a linked dataset Γ and a set of such inference rules R, pre-compute

and store the closure $\mathsf{lm}(\mathsf{merge}(\Gamma) \cup R)$,[18], issuing queries as they are input directly against the closure. Caching the full closure thus mitigates the expense of reasoning during query-time, but may conversely incur huge pre-runtime costs, storage overheads, as well as the cost of truth maintenance in dynamic scenarios.

Example 4

As an example, let us consider the following variant of Query 3: (i) instead of explicitly following the `owl:sameAs` link, we assume the necessary inferences are supplied by reasoning; (ii) we ask for all `rdfs:label`s of the company (as opposed to just `skos:prefLabel`).

QUERY 3′

```
SELECT ?D ?R ?L
WHERE { nyt:49586210195898795812 nytimes:latest_use ?D ;
                                 dbo:revenueUSD ?R ;
                                 rdfs:label ?L }
```

while the first triple pattern is matched by explicitly stated data, the subsequent query-relevant triples must be obtained from the closure with respect to, e.g., the rules in Table 3, which contains (amongst other triples):

```
nyt:49586210195898795812 nytimes:latest_use "2010-04-27"^^xsd:date ;
                         dbo:revenueUSD 1.06916E11 ;
                         rdfs:label "IBM"@en ,
                    "International Business Machines Corporation"@en .
```

leading to the following solutions:

?D	?R	?L
2010-04-27	1.06916E11	"IBM"@en
2010-04-27	1.06916E11	"International Business Machines Corporation"@en

◊

3 Overall Approaches and Challenges to Reason over and Query Linked Data

We identify two main approaches to reason and query over Linked Data:

1. Data-warehousing approaches for querying linked data are typically deployed for RDF search engines such as Sindice [54] or SWSE [40]. These engines provide query interfaces over a local centralised index of Linked Data harvested

[18] That is, all RDF triples entailed by the RDF graph obtained from Γ (read as facts) and R

from the Web and typically use rule-based materialisation (as presented in Section 2.4) to cautiously infer additional RDF triples; Section 5 will discuss such cautious reasoning techniques that are tailored to not infer too much information in this setting.

2. Rather than relying on a centralised index, Linked Data itself can be viewed as a database that can be queried directly and dynamically [33]. That is, starting from the URIs appearing in the query, or from a set of seed URIs, query-relevant data is navigated following Linked Data principles and retrieved dynamically. The main advantage of this approach is that results are much fresher and all query-relevant data do not need to be known locally. However, the main weaknesses of this approach are that performing remote lookups at query-time is costly, potentially a lot intermediate data are transferred, and the recall depends significantly on the seed URIs and conformance of the query-relevant sources with Linked Data principles. In Section 6, we will present such an approach and discuss how reasoning can be incorporated.

Getting back to the challenges enumerated in the introduction, let us now briefly discuss how these affect the architectural choice for a particular reasoning and querying infrastructure.

C1 *Linked Data is huge.* Our example contains only sample data from two example datasets in the Linked Data Web. The most recent incarnation of the Linking Open Data cloud (from September 2011), is claimed to represent over 31 billion triples spread across 295 datasets.[19] It is obvious that staying aware of and gathering this dynamically evolving data for query processing is an issue in terms of scale, but also in terms of deciding what parts of which datasets are relevant for the query at hand. For example, broad queries like Query 5, without further scope, are notoriously hard to answer, since instances of `foaf:Person` are spread over various datasets and individual RDF files spread right across the Web: while data-warehouses probably do not provide complete results on such queries, on-the-fly-traversal based approaches in the worst case can't answer such queries at all, or depending on the seed URIs, cause prohibitively many lookups during query processing.

C2 *Linked Data is not "pure" OWL.* When all usage of the `rdfs:` and `owl:` vocabulary is within the "mappable" fragment for OWL (see, e.g., Table 1), the RDF graph in question is interpretable under the OWL Direct Semantics. However, arbitrary RDF published on the Web does not necessarily fall within these bounds. For example, the FOAF vocabulary defines inverse-functional datatype properties such as `foaf:mbox_sha1sum`, which is disallowed by the restrictions under which the Direct Semantics is defined. Even worse, one may find "harmful" RDF published online that makes reasoning impossible or potentially uncontrollable, if done naively; for example, consider the triples:[20]

[19] While the LOD cloud was not updated since then, the source it is based on – `http://datahub.io/group/lodcloud` – listed 337 LOD datasets at the time of writing.

[20] A self-fulfilling prophecy: `http://axel.deri.ie/~axepol/nasty.rdf`.

```
rdfs:subPropertyOf rdfs:subPropertyOf owl:sameAs .
rdf:type rdfs:subPropertyOf owl:sameAs .
```

that could have dramatic effects when fed into a rule-based reasoner (which the reader can easily convince herself when considering our example data along with the rules from Table 3). The improper use of URIs from the special `rdf:`, `rdfs:`, and `owl:` vocabularies, i.e., the use of properties and classes from these vocabularies in assertional "non-property" or "non-class" positions is also referred to as *non-standard vocabulary use* in the literature.[21] Any reasoning approach for Linked Data has to ensure that harmful triples (be they erroneous or malicious) in published RDF are dealt with, either by being ignored or otherwise by keeping inferences "confined".

C3 *Linked Data is inconsistent.* While we have not explicitly mentioned inconsistent data in our examples – similar to non-standard use – inconsistencies often occur in published RDF data [12]. Indeed a single additional triple such as

```
dbr:Hasso_Plattner rdf:type foaf:Organization .
```

would render the example data from Fig. 2 inconsistent under OWL semantics, due to the fact that `dbr:Hasso_Plattner` is also an asserted member of the class `foaf:Person` which is declared disjoint with `foaf:Organization` in the FOAF ontology. Again, reasoning techniques need to be robust in the presence of such inconsistencies, trying to "isolate" them rather than falling for *ex falso quod libet* (anything follows from a contradiction).

C4 *Linked Data is evolving.* As per the Web itself, Linked Data is inherently dynamic [45]. For instance, our example data is not necessarily up-to-date with Linked Data currently published by NYT: for example, an article about IBM's "Watson" project was published in 2012 after the "Jeopardy" show,[22] making the `nytimes:latest_use` date of for IBM from Query $3'$ above stale. In fact, if NYT was about to update its Linked Data interface regularly with the most current articles, a query like Query $3'$ would become significantly more challenging to deal with, particularly for data-warehousing approaches that do not guarantee fresh results. We discuss these issues further in Section Section 6 below.

C5 *Linked Data needs more than RDFS and OWL.* There is more implicit knowledge hidden in Linked Data than can be captured with the semantics of RDFS and OWL alone; in fact, it may be considered quite unintuitive that Query 1 from p. 11 does not return IBM's revenue: the exchange rate between USD and EUR is itself available as data on the Web, so why shouldn't the

[21] That is, a property from these vocabularies may only be used as a predicate, and likewise classes (such as e.g. `rdfs:Resource`) may only be used as the object of `rdf:type` triples; see details within [16, 38] where non-standard vocabulary use is formally defined. Though potentially very "dangerous", non-standard triples are not always "bad": e.g., many axiomatic triples for RDFS are non-standard.

[22] Article available online at `https://www.nytimes.com/2012/01/08/jobs/building-the-watson-team-of-scientists.html` (retrieved 10 March 2013)

Web of Data be able to make use of this knowledge? However, ontology languages like OWL and RDFS do not provide means to express mathematical conversions as necessary in this example. While not solvable with current Linked Data standards and techniques, we discuss a possible approach to tackle this problem in Section 7.

4 How much OWL is needed for Linked Data?

Given the variety of combinations of techniques and RDFS/OWL profiles that can be applied for reasoning over Linked Data, an obvious question to ask is *which features of RDFS and OWL are most prominently used on the current Web of Data*?

Knowing which features are frequently used and which are infrequently used provides insights into how appropriate the coverage of various OWL profiles might be for the Linked Data use-case, and in particular, the relative costs of supporting or not supporting the semantics of a certain language primitive depending on its adoption in Web data for the setting of a given architectural choice. For example, a language feature that is costly to support or that otherwise adds complexity to a reasoning algorithm could potentially be "turned off", with minimal practical effect, if it is found to be very infrequently used in real-world data.

In this section, we thus look at the features of RDFS and OWL that are most/least widely adopted on the Web of Data. For the purposes of this study, we take the Billion Triple Challenge 2011 corpus, which consists of 2.145 billion quadruples crawled from 7.411 million RDF/XML documents through an open crawl ran in May/June 2011 spanning 791 pay-level domains.[23] This corpus represents a broad *sample* of the Web of Data. We then look into the levels of adoption of individual RDFS and OWL features within this broad corpus.

4.1 Measures Used

In order to adequately characterise the uptake of various RDF(S) and OWL features used in this corpus, we present different measures to quantify their *prevalence* and *prominence*.

First, we look at the *prevalence* of use of different features, i.e., how often they are used. Here, we must take into account the diversity of the data under analysis, where few domains account for many documents and many domains account for few documents, and so forth [38]. We thus present two simple metrics:

Doc: number of *documents* using the language feature
Dom: number of *pay-level-domains* (i.e., sites) using the language feature.

[23] A pay-level domain is a direct sub-domain of a top-level domain (TLD) or a second-level country domain (ccSLD), e.g., `dbpedia.org`, `bbc.co.uk`. This gives us our notion of "domain".

However, raw counts do not reflect the reality that the use of an OWL feature in one important ontology or vocabulary may often have greater practical impact than use in a thousand obscure documents. Thus, we also look at the *prominence* of use of different features. We use PageRank to quantify our notion of prominence: PageRank calculates a variant of the Eigenvector centrality of nodes (e.g., documents) in a graph, where taking the intuition of directed links as "positive votes", the resulting scores help characterise the relative prominence (i.e., centrality) of particular documents on the Web [55, 31].

In particular, we first rank documents in the corpus. To construct the graph, we follow Linked Data principles and consider sources as nodes, where a directed edge $(s_1, s_2) \in \mathsf{S} \times \mathsf{S}$ is extended from source s_1 to s_2 iff $\mathsf{get}(s_1)$ contains (in any triple position) a URI that dereferences to document s_2 (i.e., there exists a $u \in \mathsf{terms}(\mathsf{get}(s_1))$ such that $\mathsf{redirs}(u) = s_2$). We also prune edges to only consider (s_1, s_2) when s_1 and s_2 are non-empty sources in our corpus. We then apply a standard PageRank analysis over the resulting directed graph, using the power iteration method with ten iterations. For reasons of space, we refer the interested reader to [55] for more detail on PageRank, and to the following thesis [38] for more details on the particular algorithms used for this paper.

With PageRank scores computed for all documents in the corpus, for each RDFS and OWL language feature, we then present:

$\sum$ **Rank** the *sum of PageRank scores* for documents in which the language feature is used.

With respect to $\sum$ RANK, under the random surfer model of PageRank [55], given an agent starting from a random location and traversing documents on (our sample of) the Web of Data through randomly selected dereferenceable URIs, the $\sum$ RANK value for a feature approximates the probability with which that agent will be at a document using that feature after traversing ten links. In other words, the score indicates the likelihood of an agent, operating over the Web of Data based on dereferenceable principles, to encounter a given feature during a random walk.

The graph extracted from the corpus consists of 7.411 million nodes and 198.6 million edges. Table 4 presents the top-10 ranked documents in our corpus, which are dominated by core meta-vocabularies, documents linked therefrom, and other popular vocabularies.[24]

4.2 Survey of RDF(S)/OWL Features

Table 5 presents the results of the survey of RDF(S) and OWL usage in our corpus, where for features with non-trivial semantics, we present the measures mentioned in the previous section, as well as support for the features in various RDFS/OWL profiles. Those titled `EL`, `QL` and `RL` refer intuitively to the standard

[24] We ran another similar analysis with links to and from core RDF(S) and OWL vocabularies disabled. The results for the feature analysis remained similar. Mainly `owl:sameAs` dropped several positions in terms of the sum of PageRank.

Table 4. Top ten ranked documents

№	Document URI	Rank
1	http://www.w3.org/1999/02/22-rdf-syntax-ns	0.121
2	http://www.w3.org/2000/01/rdf-schema	0.110
3	http://dublincore.org/2010/10/11/dcelements.rdf	0.096
4	http://www.w3.org/2002/07/owl	0.078
5	http://www.w3.org/2000/01/rdf-schema-more	0.049
6	http://dublincore.org/2010/10/11/dcterms.rdf	0.036
7	http://www.w3.org/2009/08/skos-reference/skos.rdf	0.026
8	http://xmlns.com/foaf/spec/	0.023
9	http://dublincore.org/DCMI.rdf	0.021
10	http://www.w3.org/2003/g/data-view	0.017

OWL 2 profiles [25]. The RDFS standard is titled RDFS. All other profiles are non-standard proposals made in the literature. The profile titled RDFS+ refers to RDFS-Plus as proposed by Allemang and Hendler [1], which extends RDFS with lightweight OWL features. The profile titled L2 refers to a similar proposal by Fisher et al. [20] to extend RDFS with some lightweight OWL features. Description Logic Programs (DLP) was proposed by Grosof et al. [26] in order to support incomplete OWL reasoning using rules; this proposal was later built upon in Horst's pD* profile [65]. The AL profile refers to features that can be supported with rules that do not requires A-Box (i.e., assertional joins), which are expensive to compute at scale; the AL profile is used later in Section 5.

In column 'ST', we indicate which features have expressions that can be represented as a *single triple* in RDF, i.e., which features do not require auxiliary blank nodes of the form _:x or the SEQ production in Table 1 of the OWL 2 Mapping to RDF document [56]. This distinction is motivated by our initial observations that such features are typically the most widely used in Web data.

From the list of language features, we exclude rdf:type, which trivially appeared in 90.3% of documents. We present the table ordered by the sum of PageRank measure [$\sum$ RANK].

Regarding *prevalence*, we see from Table 5 that owl:sameAs is the most widely used axiom in terms of documents (1.778 million; 24%) and domains (117; 14.8%). Surprisingly (to us), RDF container membership properties (rdf:_*) are also heavily used (likely attributable to RSS 1.0 documents). Regarding *prominence*, we make the following observations:

1. The top six features are those that form the core of RDFS [53].
2. The RDF(S) declaration classes rdfs:Class, rdf:Property are used in fewer, but more prominent documents than OWL's versions owl:Class, owl:DatatypeProperty, owl:ObjectProperty.
3. owl:complementOf and owl:differentFrom are the least prominently used original OWL features.
4. Of the features new to OWL 2, owl:NamedIndividual is the most prominently used in thirty-first position. Our crawl was conducted nineteen months

Table 5. Survey of RDFS/OWL primitives used on the Web of Data and support in different tractable profiles. '∼' denotes partial support.

№	Primitive	∑ Rank	Doc	Dom	RDFS	L2	RDFS+	DLP	pD*	EL	QL	RL	AL	ST
1	rdf:Property	5.74E-01	8,049	48	✓	X	X	X	✓	X	X	X	✓	✓
2	rdfs:range	4.67E-01	44,492	89	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3	rdfs:domain	4.62E-01	43,247	89	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4	rdfs:subClassOf	4.60E-01	115,608	109	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
5	rdfs:Class	4.45E-01	19,904	43	✓	X	X	✓	✓	X	X	X	✓	✓
6	rdfs:subPropertyOf	2.35E-01	6,080	80	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
7	owl:Class	1.74E-01	302,701	111	X	X	✓	✓	✓	✓	✓	✓	✓	✓
8	owl:ObjectProperty	1.68E-01	285,412	92	X	X	✓	✓	X	✓	✓	✓	✓	✓
9	rdfs:Datatype	1.68E-01	23	9	∼	X	X	∼	∼	∼	∼	∼	∼	∼
10	owl:DatatypeProperty	1.65E-01	234,483	82	X	X	✓	✓	X	✓	✓	✓	✓	✓
11	owl:AnnotationProperty	1.60E-01	172,290	55	X	X	X	✓	X	✓	✓	✓	✓	✓
12	owl:FunctionalProperty	9.18E-02	298	34	X	X	✓	✓	✓	X	X	✓	X	✓
13	owl:equivalentProperty	8.54E-02	141	23	X	✓	✓	✓	✓	✓	✓	✓	✓	✓
14	owl:inverseOf	7.91E-02	366	43	X	✓	✓	✓	✓	X	✓	✓	✓	✓
15	owl:disjointWith	7.65E-02	230	27	X	X	X	✓	X	✓	✓	✓	X	✓
16	owl:sameAs	7.29E-02	1,778,208	117	X	✓	✓	✓	✓	✓	X	✓	∼	✓
17	owl:equivalentClass	5.24E-02	22,291	39	X	✓	✓	✓	✓	✓	✓	✓	✓	✓
18	owl:InverseFunctionalProperty	4.79E-02	111	24	X	X	✓	∼	✓	X	X	✓	✓	X
19	owl:unionOf	3.15E-02	15,162	30	X	X	X	∼	X	X	X	∼	∼	X
20	owl:SymmetricProperty	3.13E-02	120	23	X	✓	✓	✓	✓	X	✓	✓	✓	✓
21	owl:TransitiveProperty	2.98E-02	150	30	X	✓	✓	✓	✓	✓	X	✓	X	✓
22	owl:someValuesFrom	2.13E-02	1,753	15	X	X	X	∼	∼	✓	∼	∼	∼	X
23	rdf:_*	1.42E-02	293,022	62	✓	X	X	X	✓	X	X	X	✓	✓
24	owl:allValuesFrom	2.98E-03	29,084	20	X	X	X	∼	∼	X	X	∼	X	X
25	owl:minCardinality	2.43E-03	33,309	19	X	X	X	∼	X	X	X	X	X	X
26	owl:maxCardinality	2.14E-03	10,413	24	X	X	X	∼	X	X	X	∼	X	X
27	owl:cardinality	1.75E-03	3,170	24	X	X	X	∼	X	X	X	X	X	X
28	owl:oneOf	4.13E-04	74	11	X	X	X	∼	X	∼	X	∼	∼	X
29	owl:hasValue	3.91E-04	55	14	X	X	X	∼	✓	✓	X	✓	✓	X
30	owl:intersectionOf	3.37E-04	186	13	X	X	X	✓	X	✓	∼	✓	∼	X
31	owl:NamedIndividual (2)	1.63E-04	3	2	X	X	X	X	X	✓	✓	✓	✓	✓
32	owl:AllDifferent	1.55E-04	21	8	X	X	X	X	X	✓	X	✓	X	X
33	owl:propertyChainAxiom (2)	1.23E-04	14	6	X	X	X	X	X	✓	X	✓	X	X
34	owl:onDataRange	8.41E-05	3	1	X	X	X	X	X	X	X	X	X	X
35	owl:minQualifiedCardinality (2)	8.40E-05	2	1	X	X	X	X	X	X	X	X	X	X
36	owl:qualifiedCardinality (2)	4.02E-05	2	1	X	X	X	X	X	X	X	X	X	X
37	owl:AllDisjointClasses (2)	4.01E-05	2	2	X	X	X	X	X	✓	✓	✓	X	X
38	owl:maxQualifiedCardinality (2)	4.01E-05	1	1	X	X	X	X	X	X	X	∼	X	X
39	owl:ReflexiveProperty (2)	1.30E-05	2	1	X	X	X	X	X	✓	✓	X	✓	✓
40	owl:complementOf	1.96E-06	75	4	X	X	X	∼	X	X	∼	∼	X	X
41	owl:differentFrom	7.18E-07	25	7	X	X	X	✓	X	✓	X	✓	✓	✓
42	owl:onDatatype	2.72E-07	1	1	X	X	X	X	X	X	X	X	X	X
43	owl:disjointUnionOf	6.31E-08	2	2	X	X	X	X	X	X	X	X	X	X
44	owl:hasKey (2)	3.67E-08	1	1	X	X	X	X	X	✓	X	✓	X	X
45	owl:propertyDisjointWith (2)	2.43E-08	1	1	X	X	X	X	X	X	✓	✓	X	✓

Not Used: rdfs:ContainerMembershipProperty, owl:AllDisjointProperties (2), owl:Annotation (2), owl:AsymmetricProperty (2), owl:Axiom (2), owl:IrreflexiveProperty (2), owl:NegativePropertyAssertion (2), owl:datatypeComplementOf (2), owl:hasSelf (2)

after OWL 2 became a W3C Recommendation (Oct. 2009), where we note that new OWL 2 features have had little penetration in prominent Web vocabularies during that interim. Further, several OWL 2 features were not used at all in our corpus.

5. The top eighteen features are expressible with a single RDF triple. The highest ranked primitive for which this is not the case is `owl:unionOf` in nineteenth position, which requires use of RDF collections (i.e., lists). Union classes are often specified as the domain or range of a given property: the most prominent such example is the SKOS vocabulary (the seventh highest-ranked document) which specifies the range of the `skos:member` property as the union of `skos:Concept` and `skos:Container`.

In terms of profile support, we observe that RDFS has good catchment for a few of the most prominent features, but otherwise has poor coverage. Aside from syntactic/declaration features, from the top-20 features (which cover 98% of the total cumulative rank), L2 misses functional properties$_{(pos=12)}$, disjoint classes$_{(15)}$, inverse-functional properties$_{(18)}$ and union classes$_{(19)}$. RDFS-Plus omits support for disjoint$_{(15)}$ and union classes$_{(19)}$. DLP, as defined by Volz [70, §A], has coverage of all such features, but does not support inverse-functional$_{(18)}$ datatype properties. pD* does not support disjoint$_{(15)}$ or union classes$_{(19)}$.

Regarding the standard OWL 2 profiles, OWL 2 EL and OWL 2 QL both omit support for important top-20 features. Neither include functional$_{(12)}$ or inverse-functional properties$_{(18)}$, or union classes$_{(19)}$. OWL 2 EL further omits support for inverse$_{(14)}$ and symmetric properties$_{(20)}$. OWL 2 QL does not support the prevalent same-as$_{(16)}$ feature. Conversely, OWL 2 RL has much better coverage, albeit having only partial support for union classes$_{(19)}$.

Summing up, we acknowledge that such a survey cannot give a universal or definitive indication of the most important OWL features for Linked Data. First, we only survey a limited sample of the Web of Data. Second, the future may (or may not) see radical changes in how OWL is used on the Web; e.g., OWL 2 terms may soon enjoy more adoption. Still, Table 5 offers useful anecdotal insights into the extant *trends* of adoption of RDFS and OWL on the Web, and what features are the most crucial to support in a current Linked Data setting.

4.3 Survey of Datatype Use

Implementing the full range of RDF, XSD and OWL datatypes is often costly [18], with custom code (or an external library) required to support each one. We are thus interested to see which ones are most important to support.

Aside from plain literals, the RDF semantics defines a single datatype supported under RDF-entailment: `rdf:XMLLiteral` [35]. However, the RDF semantics also defines D-entailment, which provides interpretations over a datatype map that gives a mapping from lexical datatype strings into a value space. The datatype map may also impose disjointness constraints within its value space. These interpretations allow for determining which lexical strings are valid for a datatype, which different lexical strings refer to the same value and which to different values, and which sets of datatype values are disjoint from each other. An XSD-datatype map is then defined that extends the set of supported datatypes into those defined for XML Schema, including types for boolean, numeric, temporal, string and other forms of literals. Datatypes that are deemed to be am-

Table 6. Survey of standard datatypes used on the Web of Data

№	Primitive	∑ Rank	Lit	Doc	Dom	D	02
1	xsd:dateTime	4.18E-2	2,919,518	1,092,048	68	✓	✓
2	xsd:boolean	2.37E-2	75,215	41,680	22	✓	✓
3	xsd:integer	1.97E-2	1,015,235	716,904	41	✓	✓
4	xsd:string	1.90E-2	1,629,224	475,397	76	✓	✓
5	xsd:date	1.82E-2	965,647	550,257	39	✓	✗
6	xsd:long	1.63E-2	1,143,351	357,723	6	✓	✓
7	xsd:anyURI	1.61E-2	1,407,283	339,731	16	✓	✓
8	xsd:int	1.52E-2	2,061,837	400,448	31	✓	✓
9	xsd:float	9.09E-3	671,613	341,156	21	✓	✓
10	xsd:gYear	4.63E-3	212,887	159,510	12	✓	✗
11	xsd:nonNegativeInteger	3.35E-3	9,230	10,926	26	✓	✓
12	xsd:double	2.00E-3	137,908	68,682	31	✓	✓
13	xsd:decimal	1.11E-3	43,747	13,179	9	✓	✓
14	xsd:duration	6.99E-4	28,541	28,299	4	✗	✗
15	xsd:gMonthDay	5.98E-4	34,492	20,886	3	✓	✗
16	xsd:short	5.71E-4	18,064	11,643	2	✓	✓
17	rdf:XMLLiteral	4.97E-4	1,580	791	11	✓	✓
18	xsd:gMonth	2.50E-4	2,250	1,132	3	✓	✗
19	rdf:PlainLiteral	1.34E-4	109	19	2	✗	✓
20	xsd:gYearMonth	8.49E-5	6,763	3,080	5	✓	✗
21	xsd:positiveInteger	5.11E-5	1,423	1,890	2	✓	✓
22	xsd:gDay	4.26E-5	2,234	1,117	1	✓	✗
23	xsd:token	3.56E-5	2,900	1,450	1	✓	✓
24	xsd:unsignedByte	2.62E-7	66	11	1	✓	✓
25	xsd:byte	2.60E-7	58	11	1	✓	✓
26	xsd:time	8.88E-8	23	4	3	✓	✗
27	xsd:unsignedLong	6.71E-8	6	1	1	✓	✓
–	*other xsd/owl dts. not used*	—	—	—	—	—	—

biguously defined (viz. `xsd:duration`) or specific to XML (e.g., `xsd:QName`), etc. are omitted.

The original OWL specification recommends use of a similar set of datatypes to that for D-entailment, where compliant reasoners are required to support `xsd:string` and `xsd:integer`. Furthermore, OWL allows for defining enumerated datatypes.

With the standardisation of OWL 2 came two new datatypes: `owl:real` and `owl:rational`, along with novel support for `xsd:dateTimeStamp`. However, XSD datatypes relating to date, time and Gregorian calendar values are not supported. OWL 2 also introduced mechanisms for defining new datatypes by restricting facets of legacy datatypes; however, from Table 5 we note that `owl:onDatatype` (used for facet restrictions) has only very few occurrences in our corpus.

Given this broad collection of datatypes, it is interesting to see which ones are most commonly used on the Web of Data, and which ones are thus a priority to support, where we use a similar methodology as presented before. In our corpus, we found 278 different datatype URIs assigned to literals. Of these, 158 came from the DBpedia exporter which models SI units, currencies, etc., as datatypes. Using analogous measures as before, Table 6 lists the top *standard* RDF(S), OWL and XSD datatypes as used to type literals in our corpus. We omit plain literals which were used in 6.609 million documents (89%). The Lit column indicates the number of literals with that datatype. `D` indicates the datatypes supported

by D-entailment with the recommended XSD datatype map. O2 indicates the datatypes supported by OWL 2.

We make the following observations based on Table 6:

1. The top four standard datatypes are supported by both the traditional XSD datatype map and by OWL 2.
2. OWL 2 does not support `xsd:date`$_{(5)}$, `xsd:time`$_{(26)}$, or the various Gregorian datatypes$_{(10,15,18,20,22)}$.
3. Despite not being supported by any standard entailment, `xsd:duration`$_{(14)}$ was used in 28 thousand documents across four different domains.
4. Various standard datatypes are not used at all in the data. For example, `xsd:dateTimeStamp`, the "new" OWL datatypes, binary datatypes and various normalised-string/token datatypes did not appear at all.[25]

4.4 A profile of OWL for Linked Data?

Our analysis of the adoption of RDFS and OWL has shown that while some features are broadly adopted on the Web of Data, others are barely adopted at all. Thus, it would seem possible, for example, to only support some fraction of the standard OWL features while capturing support for the broad majority of axioms present on the Web of Data. In Table 5, we already saw that the most frequently used features corresponds with the ability to represent their respective axioms in RDF as a single triple (and thus without blank nodes if being interpreted under the Direct Semantics).

In previous work, we thus proposed the OWL LD (Linked Data) profile, which is a proper subset of OWL 2 RL supporting only those features that are expressible with a single RDF triple [22]. The RDF-Based Semantics of the OWL LD profile can be (partly) supported by means of a subset of the OWL 2 RL/RDF rules relating to the supported features. We also provide a grammar under which the Direct Semantics of the profile can be supported, making (optionally) conformant documents compatible with the OWL Direct Semantics. We propose that OWL LD – as a practical, terse profile – facilitates greater ease of implementation for Linked Data reasoning applications, while maintaining high coverage of commonly used features.

5 Rule-Based inference for Linked Data: Authoritative vs. Context-Dependent Reasoning

A common strategy for reasoning over multiple sources is to simply merge them together and compute the deductive closure over the resulting monolithic RDF graph. However, when dealing with arbitrary sources from the Web, one cannot expect the data to always adhere to strict rules or to be universally infallible. Web data is highly heterogeneous and unexpected usage of data and data schema

[25] In fact, `owl:real` does not have a lexical space, and thus cannot be written down; irrational numbers are difficult to write down.

is common. For example, data can be erroneous or crafted for malicious purposes. As a consequence, there are risks for a reasoner to infer undesirable logical assertions, which can be harmful for the system. These assertions increase the noise in the data collection and decrease the precision of the querying system. In addition, such inferences add an unnecessary computational overhead, which augments the demand of computational resources and limits the performance of the system. Therefore, a requirement of inference engines for Web data is the ability to cope with disparate data quality, where, in fact, incompleteness (with respect to standard RDFS/OWL profiles) is thus a desirable feature.

In this section, we present two cautious approaches for applying rule-based inferencing over diverse Linked Data in a robust manner: Context-Dependent reasoning [17] and Authoritative Reasoning [12]. Both have been designed to cope with disparate data quality and to work at large scale. However, each approach has been optimised for different scenarios. Context-Dependent reasoning is optimised for reasoning over a large number of small graphs, whereas Authoritative Reasoning is optimised for reasoning over a large single graph.

5.1 Context-Dependent Reasoning

Context-Dependent reasoning has been developed to meet the requirements of the Sindice search engine project [54]. The Sindice search engine indexes a large number of Linked Data documents, each of which contains a small RDF graph. Reasoning over these graphs enables to make explicit what would otherwise be implicit knowledge, adding value to Sindice's search engine results to ultimately be more competitive in terms of precision and recall [50].

The Context-Dependent reasoning approach has been designed to work on a multitude of small graphs in a distributed manner. Each computing node will perform the deductive closure of one graph. A *data source* is divided into small sub-graphs, e.g., on a per-entity basis or on a per-document basis such as in the Sindice search engine. Each of these graphs represents a *contextual graph*. Larger contextual graphs can be constructed from smaller ones depending on the needs. For example, one can aggregate all documents that are connected with `owl:sameAs` links into a single contextual graph if one needs to reason across `owl:sameAs` links.

A fundamental requirement in the design of the Context-Dependent reasoning approach has been to confine T-Box claims (aka., terminological claims, aka. schema claims, as per Table 3) and reasoning tasks into "contexts" in order to track the provenance of inference results. By tracking the provenance of each individual T-Box claim, we are able to prevent one ontology to alter the semantics of other ontologies on a global scale. In addition, such a context-dependent approach provides an efficient distributed computing model which scales linearly with the amount of data [17].

To reason over contexts, we assume that the ontologies that these contexts refer to are either included explicitly with `owl:imports` declarations or implicitly by using property and class URIs that dereference directly to the data describing the ontology itself. This later case should be the standard if the W3C best

practices for publishing ontologies [51] and the Linked Data principles [7] are followed by data publishers. As ontologies might refer to other ontologies, the import process then needs to be recursively iterated as explained in the next section.

A naive approach would be to execute such a recursive fetching for each contextual graph and to create an *aggregate context* [28], i.e., the *RDF merge* of the contextual graph and of the imported ontologies. At this point the deductive closure of the aggregate context can be computed. Such a naive procedure is however obviously inefficient since a lot of processing time will be used to recalculate the T-Box deductions which could be instead reused for possibly large numbers of other contextual graphs. Thus an ontology base is used to store and reuse such deductions and is described next.

Reasoning with Contexts The notions of context and lifting rules presented in the following are based on Guha's context mechanism [28]. Its aim is to control the integration of data and ultimately avoid the aggregation of data that may result in undesirable inferred assertions.

Within his framework, a *Context* is a first class resource and denotes the scope of validity of a statement. The contents of the context are said to be true in that context. This scope is defined by the symbol *ist* ("is true in context"), introduced by Guha in [27]. The notation $ist(c, \varphi)$ states that a proposition φ is true in the context c. Since contexts are first class objects, it becomes possible to define expressive formulae whose domains and ranges are contexts. An example is the so called *Lifting Rule* that enables to *lift* axioms from one context to another.

An *Aggregate Context* is a subclass of *Context*. Its content is composed by the contents lifted from other contexts. An aggregate context must contain the full specification of what it imports. In our case, each contextual graph is considered an *Aggregate Context*, since it always contains the specification of what it imports through explicit or implicit import declarations, as explained next.

Import Closure of RDF Models On the Semantic Web, ontologies are published in order to be easily reused by third parties. OWL provides the `owl:imports` primitive to indicate the inclusion of a target ontology inside an RDF model. Conceptually, importing an ontology brings the content of that ontology into the RDF model.

The `owl:imports` primitive is transitive. That is, an import declaration states that, when reasoning with an ontology O, one should consider not only the axioms of O, but the entire *import closure* of O. The *import closure* of an ontology O is the smallest set containing the axioms of O and all of the axioms from the ontologies that O (transitively) imports. For example, if ontology O_A imports O_B, and O_B imports O_C, then O_A imports both O_B and O_C.

Implicit Import Declaration Most RDF models published on the Web do not contain explicit `owl:imports` declarations. For example, among the 228 million

documents in Sindice, only 704 thousand declare at least one `owl:imports` link; also the example `dbo:` and `foaf:` ontologies[26] in our examples do not contain any explicit `owl:imports` links. Instead, many RDF models generally refer to existing ontologies by their classes or property URIs. For example, most FOAF profile documents do not explicitly import the FOAF ontology, but instead just directly use the classes and properties of the FOAF vocabulary, which dereference to the FOAF ontology. Following Linked Data principles, the URIs of the classes and properties defined in an ontology should be dereferenceable and should provide the machine-processable definition of the vocabulary (presumably given in RDFS/OWL).

That is, in the presence of dereferenceable class or property URIs, we perform what we call an *implicit import*. By dereferencing the URI, we attempt to retrieve a graph containing the description of the ontological entity identified by this URI and to include its content inside the source RDF model. The implicit import is also considered transitive.

Example 5

In Fig. 2, if a RDF model such as `dbr:Werner_von_Siemens` refers to an ontological entity such as `dbo:Person` from the ontology `dbo`, and if `dbo` refers to an ontological entity `foaf:Person` in an ontology `foaf`, then the model imports the two ontologies given by `dbo` and `foaf`. ◊

Import Lifting Rules Guha's context mechanism defines the *importsFrom* lifting rule [28] which corresponds to the inclusion of one context into another. The *owl:imports* primitive and the implicit import declaration are easily mapped to the *importsFrom* rule.

A particular case is when import relations are cyclic. Importing an ontology into itself is considered a null action, so if ontology O_A imports O_B and O_B imports O_A, then the two ontologies are considered to be equivalent [4]. Based on this definition, we extend Guha's definition to allow cycles in a graph of *importsFrom*. We introduce a new symbol eq, and the notation $eq(c_1, c_2)$ states that c_1 is equivalent to c_2, i.e., that the set of propositions true in c_1 is identical to the set of propositions true in c_2.

Definition 9 (Cyclic Import Rule). *Let c_1 and c_2 be two contexts. If c_1 contains the proposition $importsFrom(c_1, c_2)$ and c_2 the proposition $importsFrom(c_2, c_1)$, then the two contexts are considered equivalent:*

$$ist(c_2, importsFrom(c_2, c_1)) \wedge ist(c_1, importsFrom(c_1, c_2)) \rightarrow eq(c_1, c_2)$$

[26] In reality, the ontology defining the vocabulary in the `dbo` namespace is split over many documents: one per class and property term; however, this is not important for the current discussion.

Deductive Closure of RDF Models In Context-Dependent reasoning, the deductive closure of a graph is the set of assertions that are entailed in the aggregate context, composed of the graph itself and its ontology import closure. We now explain how the deductive closure of an aggregate context is performed. Given two contexts c_1 and c_2, for example a Linked Data document and an ontology, their axioms are lifted into an aggregate context labelled $c_1 \wedge c_2$. The deductive closure of the aggregate context is then computed using the rule-based inference engine.

It is to be noticed that the deductive closure of an aggregate context can lead to inferred statements that are not true in any of the source contexts alone.

Example 6

In Fig. 2, if a context c_1 contains an instance `dbr:Werner_von_Siemens` of the class `dbo:Person`, and a context c_2 contains a proposition stating that `dbo:Person` is equivalent to `foaf:Person`, then the entailed conclusion that `dbr:Werner_von_Siemens` is a `foaf:Person` is only true in the aggregate context $c_1 \wedge c_2$:

$$\begin{aligned} ist(c_1, \text{dbo:Person}(x)) \ \wedge \\ ist(c_2, equivalentClass(\text{dbo:Person}, \text{foaf:Person})) \rightarrow \\ ist(c_1 \wedge c_2, \text{foaf:Person}(x)) \end{aligned}$$

$\Diamond$

The set of inferred statements that are not true in any of the source contexts alone are called *aggregate entailments*:

Definition 10 (Aggregate Entailment). *Let c_1 and c_2 be two contexts with respectively two propositions φ_1 and φ_2, $ist(c_1, \varphi_1)$ and $ist(c_2, \varphi_2)$, and $\varphi_1 \wedge \varphi_2 \models \varphi_3$, such that $\varphi_2 \not\models \varphi_3$, $\varphi_1 \not\models \varphi_3$; then we call φ_3 a newly entailed proposition in the aggregate context $c_1 \wedge c_2$. We call the set of all newly defined propositions an aggregate entailment and denote it as Δ_{c_1,c_2}:*

$$\begin{aligned} \Delta_{c_1,c_2} = \{ & ist(c_1, \varphi_1) \wedge ist(c_2, \varphi_2) \models ist(c_1 \wedge c_2, \varphi_3) \\ & and \ \neg(ist(c_1, \varphi_3) \vee ist(c_2, \varphi_3))\} \end{aligned}$$

The aggregate entailment property enables the reasoning engine to confine inference results to specific contexts and therefore protects other contexts from unexpected data usage. Unexpected data usage in one context will not alter the intended semantics of other contexts if and only if no direct or indirect import relation exists between them.

When considering (in our case (Horn) rule-based) RDFS/OWL inferences only, aggregate contexts enjoy the following monotonicity property[27]: if the aggregate context $c_1 \subseteq c_2$ then $ist(c_2, \phi)$ implies $ist(c_1, \phi)$, or respectively, for overlapping contexts, if $ist(c_1 \cap c_2, \phi)$ implies both $ist(c_1, \phi)$ and $ist(c_2, \phi)$. This property is exploited in the implementation of the ontology base, which is described next, to avoid storing duplicate inferred statements.

Context-Dependent Ontology Base A problem when reasoning over a large number of contextual graphs independently is that the process of computing the ontology import closure and its deductive closure has to be repeated for each contextual graph. This is inefficient since the computation of the import closure and the deductive closure is resource demanding and can in fact be reused for other contextual graphs. The import closure necessitates executing multiple Web requests that place load on the network and take time, whereas the computation of the deductive closure is CPU bound. In addition, the computation of the T-Box closure is more CPU intensive than the computation of the A-Box closure [17]. This observation suggests to focus on the optimisation of the T-Box closure computation. Thanks to the smaller scale of the T-Box with respect to the A-Box, we can store the computed ontology import closure as well as the deductive closure in an *ontology base* in order to reuse them in later computation.

The ontology base, which can be seen as a persistent context-dependent T-Box, is in charge of storing any ontology discovered on the Web along with their import relations. The ontology base also stores the inference results that has been performed in order to reuse them later. The ontology base serves the inference engine by providing the appropriate and pre-computed T-Box for reasoning over a given contextual graph.

Details on the formalisation of the ontology base and of an optimised strategy to update the ontology base can be found in [17].

Implementation and Scalability The ontology base is implemented using an RDF database to store the ontology statements in their context. A secondary index is used to store the import relations between the contexts. A caching mechanism is used on top of the ontology base to cache frequent requests. The caching mechanism is especially useful when processing multiple contextual graphs from a single data source. Since contextual graphs from a same data source are likely to be described with the same ontologies, the requests to the ontology base are identical and the cache hit rate increases.

The reasoning engine that is used by the ontology base is specifically designed and optimised to compute entailments in memory using a standard bottom-up semi-naive evaluation approach. Each RDF term in a statement is mapped to a unique identifier (integer). Statements are indexed using in-memory data structures, similar to triple tables, in order to lookup any kind of statement

[27] We remark here that under the addition of possibly non-monotonic rules to the Semantic Web architecture, this context monotonicity only holds under certain circumstances [59].

patterns. Rules are then checked against the index in an iterative manner, with one rule being applied at a given iteration. The result of the rule is then added to the index before proceeding to the next iteration. Iterations continue until a fixpoint is reached. For rules that requires joins between multiple statements, since we are working with a small amount of data and a small number of elements, we rely on an efficient merge-join algorithm where both relations are sorted on the join attribute using bit arrays. The bit arrays are then intersected using bitwise operations.

The A-Box reasoning process is distributed by dividing up the large A-Box on a per-context basis. Each context provides a chunk of data that is distributed to different computing nodes. A computing node acts independently as an A-Box reasoner and has its own ontology base. The A-Box rule engine is based on the same rule engine used by the ontology base.

Since each chunk of data is relatively small, the deductive closure of the A-Box can be entirely performed in memory without relying on disk accesses. With respect to other distributed approaches that perform reasoning on the global model, we avoid reading and writing multiple times the data directly from the disk, and therefore we obtain better performance. Importantly, the distributed model scales linearly with the number of available nodes in the cluster since replicating the ontology base on each machine allows for embarrassingly parallel execution during A-Box reasoning.[28]

The Context-Dependent reasoning implementation has been in use by the Sindice search engine since 2008. The reasoner supports the pD* profile [65], though the Context-Dependent approach generalises straightforwardly to any materialisation mechanism. It is running on a Hadoop cluster of 150 computing nodes as part of the indexing pipeline of Sindice. It has enabled Sindice to reason over more than 700 million documents, which represents a total of more than 50 billion triples.

5.2 Authoritative Reasoning

The Authoritative reasoning algorithm was developed to provide RDFS and OWL materialisation support in the context of the Semantic Web Search Engine (SWSE) project [40], with similar proposals made in the context of reasoning over class hierarchies for the Falcons search engine [15]. As opposed to the Context-Dependent method, which partitions the problem of reasoning into a large collection of (relatively) small contexts, the Authoritative reasoning algorithm rather considers a single large RDF graph (in line with its SWSE use-case). Tackling the fallibility of Web data, Authoritative reasoning builds a single global T-Box that only includes axioms from "trusted sources". The core intuition of authoritative reasoning is that the T-Box axioms extracted from an ontology on the Web should only be able to affect reasoning over data that instantiates the terms in that ontology's namespace.

[28] That is, no communication is required between machines, where each can thus process their own content independently

Example 7

In the data we merge from various Web sources, assume we find the following two triples, both of which we consider to be T-Box axioms and neither of which we initially know whether to trust or not:

```
foaf:Person rdfs:subClassOf geo:SpatialThing .
foaf:Person rdfs:subClassOf ex:EvilEntity .
```

Let's take three triples instantiating the classes involved:

```
ex:Fred a foaf:Person .
ex:Jill a geo:SpatialThing .
ex:Tim a ex:EvilEntity .
```

Under RDFS semantics, these triples have the following corresponding entailments:

```
ex:Fred a geo:SpatialThing .
ex:Fred a ex:EvilEntity .
```

According to the semantics of `rdfs:subClassOf`, the original T-Box axioms only affect the inferences possible over data instantiating the `foaf:Person` class. As to whether these T-Box axioms can be trusted, we thus ask: are either of these T-Box axioms given in the document dereferenced by `foaf:Person`? The first one is indeed in the FOAF ontology, and hence can be trusted (and is considered "authoritative" along with its inferences). The second one is not, and will not be considered by the authoritative reasoning process. ◊

This intuitive notion of which sources to trust for individual T-Box axioms then relies on two prerequisites:

T-Box distinguishable from A-Box: We assume that T-Box triples in the data (and triple patterns in the rules) can be distinguished from A-Box triples.

Authoritative relation: We assume an authoritative relation that maps from an RDF document to a set of RDF terms it can speak authoritatively about.

We now discuss these general prerequisites in more detail for the setting of applying RDFS/OWL reasoning over Linked Data.

T-Box distinguishable from A-Box We discussed previously that T-Box data intuitively refers to ontological/schema definitions using the RDFS and OWL standards to define the semantics of classes and properties in a vocabulary. This intuition is sufficient for our purposes, where more precise definitions of T-Box and A-Box in the context of Authoritative reasoning are provided, e.g., in [12].

Loosely related to the notion of meta-modelling in OWL, our A-Box also contains the T-Box data (but not vice-versa). Thus, we can reason over schema triples analogous to if they were assertions. We also split the body of rules into a (possibly empty) A-Box and (possibly empty) T-Box graph pattern, where we define a T-Box triple pattern as any pattern that can only unify with a T-Box triple, and we define an A-Box triple pattern as the complement.

Example 8

Take the following OWL rule (cax-eqc1 in OWL 2 RL/RDF):

$$(?X, \mathsf{a}, ?C_2) \leftarrow \underline{(?C_1, \mathsf{owl{:}equivalentClass}, ?C_2)}\,,\ (?X, \mathsf{a}, ?C_1)$$

The first (underlined) triple pattern in the body is considered T-Box since it can only be matched by T-Box triples. The second triple pattern in the body is considered A-Box because it is not a T-Box pattern. We do not need to categorise the head of the rule in this manner. Of course, the A-Box pattern in the body may also match a T-Box triple, as per the previous meta-modelling discussion. ◊

Authoritative reasoning then involves checking the source of T-Box knowledge. Incorrect or malicious T-Box triples are the most "dangerous" in a reasoning context, where, for example, if a dataset contains millions of instances of `foaf:Person`, a single T-Box triple stated in an arbitrary location – such as one of the following

```
foaf:Person rdfs:subClassOf ex:EvilEntity .
foaf:Person rdfs:subClassOf owl:Nothing .
```

can affect inferences computed for all the millions of instances of `foaf:Person` defined in Linked Data.

Authoritative Relation Next, we need to establish a relationship between RDF sources and the set of RDF terms they speak authoritatively for. In the Linked Data setting, we can establish this authoritative relation by directly using the notion of dereferencing.

Definition 11 (Authoritative sources for terms). *We denote a mapping from a source URI to the set of terms it speaks* authoritatively *for as follows:*

$$\begin{aligned}\mathsf{auth}:\ & \mathsf{S} \to 2^{\mathsf{C}} \\ & s \mapsto \{c \in \mathsf{U} \mid \mathsf{redirs}(c) = s\} \cup \big(\mathsf{terms}(\mathsf{get}(s)) \cap \mathsf{B}\big)\end{aligned}$$

A source is thus authoritative for all URIs that dereference to it and all blank nodes it mentions. This formalises, for example, the intuitive relationship that exists between the FOAF ontology and the set of terms in the `foaf:*` namespace

that dereference to it.[29] No document is considered authoritative for literals, though this has little effect on the reasoning process.

Authoritative reasoning is applied over a Linked Dataset as given in Definition 1, which tracks the source associated with each RDF graph. Furthermore, the algorithm requires knowledge about redirects to establish the authoritative function. In practice, these data are replicated locally for the reasoning engine to access; the reasoner does not perform live lookups.

Authoritative Reasoning The primary goal of the authoritative reasoning process is to safe-guard widely used vocabularies from redefinition in arbitrary locations. Precise definitions and guarantees for authoritative reasoning are given elsewhere in [38, 12]. Here sketching the main intuition, given an ontology O providing a set of T-Box axioms and G an arbitrary RDF graph (e.g., a Web document or a merge of documents), if G does not mention any term for which O is authoritative, and O is not an implicit import of such a document, then we do not want the *T-Box axioms* provided by O to affect materialisation over G.

Thus, for example, if G instantiates vocabulary terms from the FOAF ontology but not from the DBpedia ontology, then the T-Box extracted from DBpedia should not affect inferencing over the A-Box of G. The implicit imports of the FOAF ontology can, however, affect inferencing over the T-Box of G, even if their terms are not explicitly mentioned. For example, the FOAF ontology states that `foaf:Person` is a sub-class of `geo:SpatialThing`; if G contains instances of `foaf:Person`, they will be inferred to be instances of `geo:SpatialThing` and it will then be the prerogative of the corresponding WGS84 Geo Ontology to define what inferences are possible over the latter class, even though the corresponding class is not explicitly referenced by G.

Whether or not a T-Box axiom is considered authoritative then directly depends on the rules being applied in the reasoning process. In fact, a T-Box axiom may be authoritative with respect to one rule and not another.

Example 9

Take the following T-Box triple from Fig. 1:

```
dbo:Person owl:equivalentClass foaf:Person .
```

This triple is given by the document that `dbo:Person` dereferences to. If we then take OWL 2 RL/RDF rule cax-eqc1 mentioned in Example 8:

$$(?X, \mathsf{a}, ?C_2) \leftarrow \underline{(?C_1, \mathsf{owl{:}equivalentClass}, ?C_2)}\,,\ (?X, \mathsf{a}, ?C_1)$$

the T-Box triple is authoritative for this rule since it translates data about `dbo:Person` instances into `foaf:Person` instances.

[29] The source URI will often not share the namespace of the URIs it is authoritative for since redirects (esp. PURLS) are commonly used for dereferencing schemes.

Now, if we take the same T-Box triple but instead take OWL 2 RL/RDF rule cax-eqc2:

$$(?X, \mathsf{a}, ?C_1) \leftarrow \underline{(?C_1, \mathsf{owl{:}equivalentClass}, ?C_2)}\,,\ (?X, \mathsf{a}, ?C_2)$$

the T-Box triple is no longer authoritative since it translates instance data about `foaf:Person` into `dbo:Person` instances, and as justified before, we do not want the DBpedia ontology document to be able to affect inferences over (FOAF) data that do not contain any DBpedia terms for which the document is authoritative. ◊

Thus, we see that T-Box axioms are only authoritative with respect to the rule(set) under consideration. When applying rules over the data, we can then apply a relatively straightforward (and slightly stricter) condition to ensure that the T-Box axiom matched in the body of the rule will lead to an authoritative inference. Recall that the document serving the T-Box axiom should be authoritative for at least one term mentioned in the A-Box being reasoned over. We thus look at the terms bound to variables that appear in both the T-Box and A-Box part of the rule body. For a given rule application, if the document providing the T-Box axiom is authoritative for at least one such term, we deem the rule application to be authoritative; otherwise we consider it to be non-authoritative.

Example 10

From the previous example, if we look at rule cax-eqc1, the only variable common to the T-Box and A-Box segments of the rule body is $?C_1$. Taking the given T-Box axiom, $?C_1$ is bound to `dbo:Person` for which the T-Box source is authoritative. Hence, for any triple of the form $(?X, \mathsf{a}, \mathsf{dbo{:}Person})$ in our A-Box data, we can authoritatively infer the corresponding triple of the form $(?X, \mathsf{a}, \mathsf{foaf{:}Person})$.

Instead taking rule cax-eqc2, the only variable common to the T-Box and A-Box segments of the rule body is $?C_2$. For the given T-Box axiom, $?C_2$ is bound to `foaf:Person` for which the DBpedia ontology is not authoritative. Hence, for any A-Box triple of the form $(?X, \mathsf{a}, \mathsf{foaf{:}Person})$ in our data, authoritative reasoning will block the inference of $(?X, \mathsf{a}, \mathsf{dbo{:}Person})$ (unless the T-Box axiom is also given in the FOAF ontology, which in reality it is not). ◊

If a rule contains only T-Box or only A-Box patterns in its body, authoritative reasoning does not block the inferences. Any inferences from T-Box level reasoning are assigned to the context of the source from which *all* of the premises originate; if a T-Box level inference involves premises from multiple documents,

it is not considered to have any fixed source and can never be authoritative.[30] Standard Authoritative reasoning does not affect rules consisting of only A-Box patterns, which includes rules that provide `owl:sameAs` entailments over instances (see Table 3).

Implementation and Scalability Unlike the Context-Dependent reasoning algorithm, Authoritative reasoning does not partition the problem of reasoning into small contexts. Instead, Authoritative reasoning requires applying inference over the entire dataset in "one go". Thus, the methods of inference applied must scale to the entire dataset (and not just individual contexts). We implement such methods in the Scalable Authoritative OWL Reasoner (SAOR) [41], designed to apply lightweight OWL reasoning over large collections of diverse Linked Data. The reasoning process is divided into two distinct phases, as follows:

Compute T-Box The T-Box is extracted from the main body of data and axioms are analysed for authoritativeness with respect to the given ruleset. If required, T-Box level reasoning is applied.

Reason over A-Box The A-Box is reasoned over with respect to the global authoritative T-Box built in the previous phase.

In terms of scalability, when dealing with large collections of Linked Data, we observe that the T-Box is generally quite small (e.g., typically $< 0.1\%$ of the total triple count [41]) and is frequently accessed during the reasoning process; hence we load the T-Box into memory such that it can be accessed efficiently. Furthermore, a variety of papers have demonstrated that splitting the T-Box from the main body of data allows for effective distributed computation of materialised inferences [72, 69, 41, 68], where (similar to Context-Dependent reasoning) the T-Box is replicated to each machine performing inference. Indeed, if the ruleset does not contain any rules with more than one A-Box pattern in the body (as is the case for, e.g., the RDFS inference rules [cf. Table 3] and for rules cax-sco, cax-eqc1 and cax-eqc2 introduced previously in the examples), then this form of distributed materialisation can reason over the A-Box in an embarrassingly parallel fashion for any arbitrary distributed partitioning of the A-Box data. Rules with multiple A-Box patterns in the body require joins over the very large A-Box (typically between machines), and in many cases, such rules can produce huge volumes of materialisations; for example, transitive-property reasoning is quadratic with respect to the extension of that property in the A-Box.

An important practical question then is how much is lost by not considering rules that have multiple A-Box patterns in the body? In Table 5, the `AL` profile lists the features that can be supporting using "A-Linear rules": rules with only one assertional pattern [41]. In SAOR, we implement A-Linear OWL

[30] In any case, informally, we can conjecture that terminology-specific reasoning in rulesets such as OWL 2 RL/RDF is (often) redundant with respect to assertional inferencing applied recursively; for example, performing the transitive closure of sub-class relations is only necessary to infer sub-class relations, where recursive application of cax-sco will infer all assertions without it.

2 RL/RDF rules: the intersection of AL and RL. One of the most prominent features we lose is the ability to reason over owl:sameAs relations; both to infer such relations through, e.g., functional properties and inverse-functional properties, and to support the semantics of equality as per the rules in Table 3 (only eq-sym is A-Linear).

In terms of completeness with respect to standard bottom-up rule-evaluation (i.e., without any distinction between T-Box or A-Box), the main limitation of considering a separate static T-Box while reasoning over the A-Box is that it can lead to incompleteness if new T-Box triples are found while reasoning over the A-Box [41] (these triples will not be reflected in the T-Box). Inference of T-Box triples during A-Box reasoning can occur due to non-standard use of the core RDFS or OWL vocabulary (see Section 3). Workarounds for this problem are possible: for example to recursively identify and reason over non-standard triples in a pre-processing step, etc. However, non-standard use of the RDF(S) and OWL vocabulary is not found often in Linked Data, with notable exceptions being, e.g., the RDFS axiomatic triples and the documents dereferenced by the RDF, RDFS and OWL terms themselves.

In the SAOR system, following previous papers [72, 69], we also perform A-Box reasoning in a distributed setting. We have evaluated the applicability of SAOR over 1 billion Linked Data triples taken from 4 million Linked Data documents. Using a variety of optimisations for our A-Linear profile of OWL 2 RL/RDF, on a cluster of nine machines with 4GB of RAM and 2.2 GHz single-core processors, we computed 1 billion unique and authoritative inferences in about 3.5 hours [41], roughly doubling the input size. Without considering the authority of inferences, we estimated that the volume of materialisation would increase by 55×, even for the lightweight reasoning profile being considered [12].

5.3 Comparison and Open Issues

Meeting the Challenges Tackling C1 (scalability) in the list of challenges enumerated in Section 3, both Context-Dependent reasoning and Authoritative reasoning use distributed computing and partitioning techniques and various rule-based optimisations to enable high levels of scale.

Tackling C2 (impure and fallible OWL), both approaches analyse the source of input axioms and apply cautious materialisation, where incompleteness with respect to standard OWL profiles is thus a feature, not a "bug".[31] Both approaches can use rule-based inferencing to support an incomplete RDF-Based semantics, which does not require input graphs to conform to OWL 2 DL restrictions enforced by OWL's Direct Semantics.

Regarding C3 (inconsistencies), both approaches use monotonic rule-based reasoning techniques that do not reduce the deductive reasoning process to unsatisfiability checking, and thus do not fall into "ex falso quod libet". Inconsistencies can be ignored. However, in the case of SAOR, we have also looked at

[31] Importantly, a *non-standard* version of completeness can be rigorously defined in both cases. See, e.g., [41] for details in the SAOR case.

resolving the contradictions presented by inconsistencies: we investigated using an annotated logic program framework to rank assertions under a PageRank model, where the marginal assertion in a contradiction is defeated [12].

With respect to C4 (dynamic Linked Data), Context-Dependent reasoning allows entailments to be updated on a context-by-context basis, where changes to the ontology base can also be efficiently supported (see [17]); Authoritative reasoning does not directly support incremental updates, where truth maintenance techniques would be required. (The following section presents an approach that better handles reasoning and querying over Linked Data in highly dynamic scenarios.)

With respect to C5 (more than RDFS/OWL required), both approaches generalise to the application of arbitrary rule-based reasoning, where the Context-Dependent framework – a means to manage contexts – generalises further to any form of deductive (or even inductive) reasoning process, as required.

Comparison of Both Approaches In terms of the differences between both approaches, the Context-Dependent approach is designed to run over small contexts, typically involving one "assertional" document and its recursive ontology imports. Although the framework can handle aggregate contexts, the larger these aggregate contexts become, the closer Context-Dependent reasoning resembles the naïve case of standard reasoning over a monolithic graph. Thus, Context-Dependent reasoning is not well-suited to deriving entailments across assertional documents. The T-Box generated during Authoritative reasoning can be used to cautiously derive entailments across assertional documents (effectively reflecting a common consensus for a T-Box across all contexts); however, in practice, to achieve scalability, the A-Linear profile disables such inferences.

Conversely, Context-Dependent reasoning trusts all axioms in a local context, whereas Authoritative reasoning does not. In other words, Context-Dependent reasoning allows non-authoritative reasoning within contexts, which Authoritative reasoning never allows. With reference to Example 9, if a document imports the DBpedia ontology involved, Context-Dependent reasoning will permit translating `foaf:Person` instances into `dbo:Person` instances, whereas Authoritative reasoning will not.

Support for same-as? A primary limitation common to both approaches is the inability to effectively reason over `owl:sameAs` relations. Context-Dependent reasoning can only process such relations with a single context, which will miss the bulk of equivalence relations between assertional documents. In theory, Authoritative reasoning can support `owl:sameAs` inferences, but for scalability reasons, rules with A-Box joins are disabled in the SAOR implementation. However, in other more focussed works, we have looked at specialised methods for authoritative reasoning of `owl:sameAs` relations in a Linked Data setting [42].

Indeed, `owl:sameAs` can produce huge volumes of inferences: in previous work [42], we found 33,052 equivalent terms within a single (correct) `owl:sameAs` clique, which would require $33,052^2 = 1,092,434,704$ triples just to materialise the pair-wise and reflexive `owl:sameAs` relations between terms in this one

group, even before applying any of the eq-rep-* rules for replacement. Given the importance of owl:sameAs reasoning for aligning entities in Linked Data, the potential expense of such reasoning, and given that equivalence relations cannot be universally trusted on the Web [29], a number of works have tackled this issue with specialised techniques and optimisations [42, 44, 68]. For example, most systems supporting owl:sameAs reasoning at large scale use a single canonical identifier to represent each set of equivalent identifiers, avoiding the explosion of data that could otherwise occur [39, 42, 68, 11]. In previous work, we applied authoritative reasoning to compute owl:sameAs relations from functional and inverse-functional properties and cardinality restrictions [42].

Interestingly, Hu et al. [42] investigate a notion of authority for owl:sameAs inferencing, assigning a level of trust to such a relation based on whether the given document is authoritative for the subject or object or both of a same-as relation (here applying authority on an A-Box level). In any case, we note that owl:sameAs is an important reasoning feature in the context of Linked Data, but similarly requires specialised techniques – that go beyond a generic reasoning framework – to handle effectively in scalable, real-world settings.

6 Enriching Link-traversal based Querying of Linked Data by Reasoning

As discussed previously, data-warehousing approaches – such as those introduced in the previous section – are not well suited for reasoning and querying over highly dynamic Linked Data. Content replicated in local indexes will quickly become out-of-date with respect to the current version of the respective sources on the Web. However, we referred in the introduction to the vision of the Web of Data itself as being a giant database spanning the Web, where certain types of queries can be posed and executed directly over the sources it contains. Such an approach for executing SPARQL queries directly over the Web of Data – called Link Traversal Based Query Execution (LTBQE) – was first proposed by Hartig et al. [33] (§ 6.1). However, the original approach did not offer any reasoning capabilities; indeed, no existing reasoning approaches at the time would seem suitable for such a scenario.

In this section, we first describe the LTBQE algorithm (a comprehensive study of the semantics and computability of LTBQE has been covered in [32]), complete with formal definitions and illustrative examples, motivate why RDF-S/OWL reasoning is useful in such a setting, and then discuss methods we have ourselves since proposed to support such reasoning features.

6.1 Overview of Baseline LTBQE

Given a SPARQL query, the core operation of LTBQE is to identify and retrieve a focused set of query-relevant RDF documents from the Web of Data from which answers can be extracted. The approach begins by dereferencing URIs found in the query itself. The documents that are returned are parsed, and

triples matching patterns of the query are processed; the URIs in these triples are also dereferenced to look for further information, and so forth. The process is recursive up to a fixpoint wherein no new query-relevant sources are found. New answers for the query can be computed on-the-fly as new sources arrive. We now formally define the key notion of query-relevant documents in the context of LTBQE, and give an indication as to how these documents are derived. [32]

Definition 12 (Query Relevant Sources & Answers). *First let* $\mathsf{uris}(\mu) := \{u \in \mathsf{U} \mid \exists v \text{ s.t. } (v,u) \in \mu\}$ *denote the set of URIs in a solution mapping* μ*. Given a query* Q *and an intermediate dataset* Γ*, we define the function* qrel*, which extracts from* Γ *a set of URIs that can (potentially) be dereferenced to find further sources deemed relevant for* Q*:*

$$\mathsf{qrel}(Q,\Gamma) := \bigcup_{tp \in Q} \bigcup_{\mu \in [\![\{tp\}]\!]_\Gamma} \mathsf{uris}(\mu)$$

To begin the recursive process of finding query-relevant sources, LTBQE takes URIs in the query—denoted with $U_Q := \mathsf{terms}(Q) \cap \mathsf{U}$*—as "seeds", and builds an initial dataset by dereferencing these URIs:* $\Gamma_0^Q := \mathsf{derefs}(U_Q)$*. Thereafter, for* $i \in \mathbb{N}$*, define:*[33]

$$\Gamma_{i+1}^Q := \mathsf{derefs}\big(\mathsf{qrel}(Q, \Gamma_i^Q)\big) \cup \Gamma_i^Q$$

The set of LTBQE query relevant sources *for* Q *is given as the least* n *such that* $\Gamma_n^Q = \Gamma_{n+1}^Q$*, denoted simply* Γ^Q*. The set of* LTBQE query answers *for* Q *is given as* $[\![Q]\!]_{\Gamma^Q}$*, or simply denoted* $[\![Q]\!]$*.*

Example 11

We illustrate this core concept of LTBQE query-relevant sources with a simple example based on Fig. 2. Let us consider our example Query 3.

First, the process extracts all raw query URIs:

$U_Q = \{\texttt{nyt:4958--}, \texttt{nytimes:latest\_use}, \texttt{owl:sameAs}, \texttt{dbo:revenueUSD}\}$

and the engine dereferences these URIs. Second, LTBQE looks to extract additional query relevant URIs by seeing if any query patterns are matched in the current dataset. LTBQE repeats the above process until no new sources are found. When no other query-relevant URIs are found, a fixpoint is reached and the process terminates with the results given over the retrieved "query-relevant documents". ◊

[32] This is similar in principle to the generic notion of reachability introduced previously [34, 32], but relies here on concrete HTTP specific operations.

[33] In practice, URIs need only be dereferenced once; *i.e.*, only URIs in $\mathsf{qrel}(Q, \Gamma_i^Q) \setminus (\mathsf{qrel}(Q, \Gamma_{i-1}^Q) \cup U_Q)$ need be dereferenced at each stage.

6.2 (In)completeness of LTBQE

An open question is the *decidability* of collecting query-relevant sources: does it always terminate? This is dependent on whether one considers the Web of Data to be infinite or finite. For an infinite Web of Data, this process is indeed undecidable [32]. To illustrate this case, Hartig [32] uses the example of a Linked Data server describing all natural numbers[34], where each $n \in \mathbb{N}$ is given a dereferenceable URI, each n has a link to $n+1$ with the predicate `ex:next`, and a query with the pattern "`?n ex:next ?np1 .`" is given. In this case, the traversal of query-relevant sources will span the set of all natural numbers. However, if the (potential) Web of Data is finite, then LTBQE is decidable; in theory, it will terminate after processing all sources. The question of whether the Web (of Data) is infinite or not comes down to whether the set of URIs is infinite or not: though they may be infinite in theory [8] (individual URIs have no upper bound for length), they are finite in practice (machines can only process URIs up to some fixed length).[35]

Of course, this is a somewhat academic distinction. In practice, the Web of Data is sufficiently large that LTBQE may end up traversing an infeasibly large number of documents before terminating. A simple worst case would be a query with an "open pattern" consisting of three variables.

Example 12

The following query asks for data on the founders of `dbr:SAP_AG`:

```
SELECT ?s ?p ?o WHERE {
  dbr:SAP_AG dbo:foundedBy ?s .
  ?s ?p ?o .
}
```

The first query-relevant sources will be identified as the documents dereferenced from `dbr:SAP_AG` and `dbo:foundedBy`. Thereafter, all triples in these documents will match the open pattern, and thus all URIs in these documents will be considered as potential query-relevant links. This will continue recursively, crawling the entire Web of Data. Of course, this problem does not occur only for open patterns. One could also consider the following query which asks for the friends of the founders of `dbr:SAP_AG`:

```
SELECT ?o WHERE {
  dbr:SAP_AG dbo:foundedBy ?s .
  ?s foaf:knows ?o .
}
```

[34] Such a server has been made available by Vrandečíc *et al.* [71], but unfortunately stops just shy of a billion. See, *e.g.*, `http://km.aifb.kit.edu/projects/numbers/web/n42`.

[35] It is not clear if URIs are (theoretically) finite strings. If so, they are countable [32].

This would end up crawling the connected Web of FOAF documents, as are linked together by dereferenceable `foaf:knows` links. ◊

Partly addressing this problem, Hartig *et al.* [33] defined an iterator-based execution model for LTBQE, which rather approximates the answers provided by Definition 12. This execution model defines an ordering of triple patterns in the query, similar to standard nested-loop join evaluation. The most selective patterns (those expected to return the fewest bindings) are executed first and initial bindings are propagated to bindings further up the tree. Crucially, later triple patterns are partially bound when looking for query-relevant sources. Thus, taking the previous example, the pattern "`?s foaf:knows ?o .`" will never be used to find query-relevant sources, but rather partially-bound patterns like "`dbr:Werner_Von_Siemens foaf:knows ?o .`" will be used. As such, instead of retrieving all possible query-relevant sources, the iterator-based execution model uses interim results to apply a more focused traversal of the Web of Data. This also makes the iterator-based implementation order-dependent: results may vary depending on which patterns are executed first and thus answers may be missed. However, it does solve the problem of traversing too many sources when low-selectivity patterns are present in the query.

Whether defined in an order-dependent or order-independent fashion, LTBQE will often not return complete answers with respect to the Web of Data [32]. We now enumerate some of the potential reasons LTBQE can miss answers.

Example 13

No dereferenceable query URIs: The LTBQE approach cannot return results in cases where the query does not contain dereferenceable URIs. For example, consider posing the following query against Fig. 2:

```
SELECT ?s ?p WHERE {
  ?s ?p nytimes:nytd_org .
}
```

As previously explained, the URI `nytimes:nytd_org` is not dereferenceable ($\mathsf{deref}(\texttt{nytimes:nytd\_org}) = \emptyset$) and thus, the query processor cannot compute and select relevant sources from interim results. ◊

Example 14

Unconnected query-relevant documents: Similar to the previous case of reachability, the number of results might be affected if query relevant documents cannot be reached. This is the case if answers are "connected" by literals, blank-nodes or non-dereferenceable URIs. In such situations, the

query engine cannot discover and dereference further query relevant data. The following query illustrates such a case:

```
SELECT ?comp ?name WHERE {
   dbr:SAP_AG foaf:name ?name .
   ?comp skos:prefLabel ?name .
}
```

Answers (other than `dbr:SAP_AG`) cannot be reached from the starting URI `dbr:SAP_AG` because the relevant documents are connected together by the literal `"SAP AG"`, which cannot be traversed as a HTTP link. ◊

Example 15

Dereferencing partial information: In the general case, the effectiveness of LTBQE is heavily dependent on the amount of data returned by the $\mathsf{deref}(u)$ function. In an ideal case, dereferencing a URI u would return all triples mentioning u on the Web of Data. However, this is not always the case; for example:

```
SELECT ?s WHERE {
  ?s owl:sameAs dbr:SAP_AG .
}
```

This quite simple query cannot be answered by link-traversal techniques since the triple "`nyt:75293219995342479362 owl:sameAs dbr:SAP_AG .`" is not accessible by dereferencing `dbr:SAP_AG` or `owl:sameAs`. ◊

The assumption that all RDF available on the Web of Data about a URI u can be collected by dereferencing u is clearly idealised; hence, later in Section 6.4 we will empirically analyse how much the assumption holds in practice, giving insights into the potential recall of LTBQE on an infrastructural level.

6.3 LiDaQ: Extending LTBQE with Reasoning

Partly addressing some of the shortcomings of the LTBQE approach in terms of completeness (or, perhaps more fittingly, *recall*), Hartig *et al.* [33] proposed an extension of the set of query relevant sources to consider `rdfs:seeAlso` links, which sometimes overcomes the issue of URIs not being dereferenceable (as per `nytimes:nytd_org` in our example).

On top of this extension, we previously proposed a system called "LiDaQ" that extends the baseline LTBQE approach with components that leverage lightweight RDFS and `owl:sameAs` reasoning in order to improve recall. Formal

definitions of the extensions we propose are available in our paper describing LiDaQ [67]. Here we rather sketch our proposals and provide intuitive examples.

Considering `owl:sameAs` links and inferences First, we propose following `owl:sameAs` links, which, in a Linked Data environment, are used to state that more information about the given resource can be found elsewhere under the target URI. Thus, to fully leverage `owl:sameAs` information, we first propose to follow relevant `owl:sameAs` links when gathering query-relevant sources and subsequently apply `owl:sameAs` reasoning, which supports the semantics of *replacement* for equality, meaning that information about equivalent resources is mapped to all available identifiers and made available for query answering. We illustrate the need for such an extension with the following example:

Example 16

Consider the following query asking for the revenue(s) of the company identified by the URI `nyt:75293219995342479362`.

```
SELECT ?rev WHERE {
  nyt:75293219995342479362 dbo:revenueEUR ?rev .
}
```

When applying this query over the data in Fig. 2, the `owl:sameAs` relationship between `nyt:75293219995342479362` and `dbr:SAP_AG` states that both URIs are equivalent and referring to the same real world entity, and hence that the information for one applies to the other. Hence, the revenue associated with `dbr:SAP_AG` should be returned as an answer according to OWL semantics. However, the baseline LTBQE approach will not return any answers since such equality relations are not considered. In summary, to answer this query, LTBQE must be extended to follow `owl:sameAs` links and apply reasoning to materialise inferences with respect to the semantics of replacement. ◊

To return answers for such examples, LTBQE needs to be extended to follow `owl:sameAs` links and apply reasoning. Thus, the set of query-relevant sources is extended to also consider documents dereferenced by looking up URIs that are equivalent to query-relevant URIs (such as `dbr:SAP_AG` in the previous example) and subsequently applying the *Same-As* subset of OWL 2 RL/RDF rules given in Table 3 over the merge of data, which performs replacement for equivalent terms related through `owl:sameAs`.

Considering RDFS inferences Second, we can extend LTBQE to consider some lightweight RDFS reasoning, which takes schema-level information from pertinent vocabularies and ontologies that describe the semantics of class and

property terms used in the query-relevant data and uses it to infer new knowledge. We motivate the need for RDFS reasoning with another straightforward example:

Example 17

Consider the following query asking for the label(s) of the company identified by the URI `dbr:IBM`.

```
SELECT ?label WHERE {
  dbr:IBM rdfs:label ?label .
}
```

When applying this query over the data in Fig. 2, baseline LTBQE will return the answer "`IBM`". However, from the schema data (right-hand side of the example), we can see that the `foaf:name` property is defined in RDFS as a sub-property of `rdfs:label`. Hence, under RDFS semantics, we should also get "`International Business Machines Corporation`" as an additional answer. As such, considering RDFS inferencing can help find more answers under the LTBQE approach. ◊

Thus we can extend LTBQE to apply further reasoning and generate more answers. Although our LiDaQ proposal only considers the RDFS rules enumerated in Table 3, the approach can be generalised to other more comprehensive rule-sets, such as OWL 2 RL/RDF.

As a first step, we must make (RDF) schema data available to the query engine, where we propose three mechanisms:

1. a static collection of schema data are made available as input to the engine (e.g., the schema data from Fig. 2 are made available offline to the engine);
2. the properties and classes mentioned in the query-relevant sources are dereferenced to dynamically build a direct collection of schema data (e.g., since mentioned in the `dbr:IBM` query-relevant document, `foaf:name` is dereferenced to get the schema data at runtime); and
3. the direct collection of dynamic schema data is expanded by recursively following links on a schema level (e.g., not only is `foaf:name` dereferenced, but `rdfs:label` is also recursively dereferenced from that document).

The first approach follows the proposals for Authoritative reasoning laid out in the previous section, whereas the latter two approaches follow a "live" version of proposals for Context-Dependent reasoning, where the second mechanism includes direct implicit imports and the third mechanism includes recursive implicit imports (as argued in that section, since `owl:imports` is quite rare within a Linked Data setting, we omit its support for brevity).

Using the schema data collected by one of these methods, in the second step, we apply rule-based RDFS reasoning to materialise inferences and make them

available for query-answering. Thus we can achieve the types of answers missing from the example.

6.4 Benefit of LTBQE Reasoning Extensions

Taken together, the two proposed reasoning extensions for LTBQE allow any client with a Web connection to answer queries, such as given in Example 4, and retrieve a full set of answers fresh from the original sources, potentially spanning multiple domains. The client does not need to index the sources in question.

With respect to generalising the benefits of reasoning, we now briefly summarise results of an empirical study we conducted to examine how LTBQE and its extensions can be expected to perform in practice; details can be found in [67]. The study took a large crawl of the Web of Data (the BTC'11 corpus) as a sample and surveyed the ratio of all triples mentioning a URI in our corpus against those returned in the dereferenceable document of that URI; this is done for different triple positions. In addition, the study also looks at the comparative amount of raw data about individual resources considering (1) explicit, dereferenceable information; (2) including `rdfs:seeAlso` links [33]; (3) including `owl:sameAs` links and inferences; (4) including RDFS inferences with respect to a static schema.

The study reports that, in the general case, LTBQE works best when a subject URI is provided in a query-pattern, works adequately when only (non-class) object URIs are provided, but works poorly when it must rely on property URIs bound to the predicate position or class URIs bound to the object position. Furthermore, we found that `rdfs:seeAlso` links are not so common (found in 2% of cases) and do not significantly extend the raw data made available to LTBQE for query-answering. Conversely, `owl:sameAs` links are a bit more common (found in 16% of cases) and can increase the raw data made available for answering queries significantly (2.5×). Furthermore, RDFS reasoning often (81% of the time) increases the amount of available raw data by a significant amount (1.8×).

We also tested the effect of these extensions for running queries live over Linked Data. We generated a large set of queries intended to be answerable using LTBQE by means of random walks across dereferenceable URIs available in our crawl. We then enabled and disabled the various configurations of the extensions proposed and ran the queries live over Linked Data sources on the Web. Summarising the results, we found that adding reasoning support to LTBQE allows for finding additional answers when directly querying Linked Data, but also introduces significant overhead. In particular, proposals to dynamically traverse implicit imports at runtime generate a huge overhead. Our conclusions were that reasoning was possible in such a setting, but is only practicable for simple queries involving few documents. Again, an excellent example of the type of query well-supported by such an approach is Query 3′ listed earlier.

7 Extending Query Rewriting Techniques by Attribute Equations for Linked Data

In this section, we are getting back to challenge C5 from the Introduction, that is, the claim that *Linked Data needs more than RDFS and OWL*. To justify this position, we return to our running example.

Example 18

We already pointed to the example of Query 1 from p. 11, where it may be considered quite unintuitive that IBM's revenue is not returned. Given the exchange rate between EUR and USD (1 EUR = 1.30 USD as of 25 March, 2013), the value for `dbo:revenueEUR` should be computable from a value for `dbo:revenueUSD` and vice versa. In general, many numerical properties are related, not by `rdfs:subPropertyOf` relations or anything expressible in RDFS/OWL, but rather by the simple mathematical equations such as, for instance:

$$revenueUSD = revenueEUR * 1.3 \quad (1)$$

$$profitEUR = revenueEUR - totalExpensesEUR \quad (2)$$

◊

While such equations are not expressible in RDFS or OWL itself, lots of emerging Linked Data is composed of interdependent numerical properties assigned to resources. Lots of implicit information would be expressible in the form of such simple mathematical equations modelling these interdependencies. These dependencies include simple conversions, e.g., between currencies as in (1), or functional dependencies between multiple properties, such as exemplified in (2).

In this section we present an approach to extend RDFS and OWL by so-called attribute equations as part of the terminological knowledge in order to enable inclusion of additional numerical knowledge in the reasoning processes for integrating Linked Data. While an exhaustive discussion of the idea of attribute equations in all depth is beyond the scope of this paper, we refer the interested reader to [10] for more details.

7.1 Extending ontologies by attribute equations

Attribute equations in [10] allow a very restricted form of simple numerical equations in multiple variables as follows.

Definition 13. *Let* $\{x_1, \ldots, x_n\}$ *be a set of variables. A* simple equation E *is an algebraic equation of the form* $x_1 = f(x_2, \ldots, x_n)$ *such that* $f(x_2, \ldots, x_n)$ *is an arithmetic expression over numerical constants and variables* $x_2, \ldots, x_n$

where f uses the elementary algebraic operators $+$, $-$, $\cdot$, $\div$ and contains each x_i exactly once. $\mathsf{vars}(E)$ *is the set of variables* $\{x_1, \ldots, x_n\}$ *appearing in* E.

That is, we allow non-polynomials for f – since divisions are permitted – but do not allow exponents (different from ± 1) for any variable; the idea here is that such equations can be solved uniquely for each x_i by only applying elementary transformations, assuming that all x_j for $j \neq i$ are known: i.e., for each x_i, such that $2 \leq i \leq n$, an equivalent equation E' of the form $x_i = f'(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ is uniquely determined. Note that since each variable occurs only once and standard procedure for solving single variable equations can be used, we write $\mathrm{solve}(x_1 = f(x_2, \ldots, x_n), x_i)$ to denote E'.[36]

In order to enable semantic support within OWL and RDFS for such simple equations where attribute URIs will be used as variable names, we need a syntactic representation. To this end, in addition to DL axioms encoded in the `rdfs:` and `owl:` vocabularies, we propose a new property (e.g., extending the `rdfs:` vocabulary) `rdfs:definedByEquation` to encode so called equation axioms. That is, we extend axioms as in Table 1 as follows:

Table 7. Mapping equation axioms to RDF

DL	RDFS
$P_0 = f(P_1, \ldots, P_n)$	P_0 rdfs:definedByEquation "$\mathsf{f}(\mathsf{P}_1, \ldots, \mathsf{P}_n)$"

Here, $P_1, \ldots, P_n$ are URIs of numerical properties (which we also call *attributes*); we write the respective arithmetic expressions $f(P_1, \ldots, P_n)$ as plain literals in this terse encoding (instead of, e.g., breaking down the arithmetic expressions into RDF triples).

Example 19

The RDF encodings of (1) and (2) are

```
dbo:revenueUSD rdfs:definedByEquation "dbo:revenueEUR * 1.3" .
dbo:profitEUR rdfs:definedByEquation
                    "dbo:revenueEUR - dbo:totalExpensesEUR" .
```

◊

As mentioned before in the context of Definition 13, we consider equations that result from just applying elementary transformations as equivalent. In order

[36] ... with analogy to notation used by computer algebra systems (such as Mathematica, cf. `http://www.wolfram.com/mathematica/`, or Maxima, cf. `http://maxima.sourceforge.net`).

to define the semantics of equation axioms accordingly, we will make use of the following definition.

Definition 14. *Let* $E\colon P_0 = f(P_1, \ldots, P_n)$ *be an equation axiom then, for any* P_i *with* $0 \leq i \leq n$ *we call the equation axiom* $\text{solve}(E, P_i)$ *the* P_i-variant *of* E*.*

As for the details on the formal semantics of attribute equations we refer the reader again to [10] and directly jump to two potential ways to implement reasoning and querying with such equations.

In principle, there are the same two potential ways to implement reasoning with property equations as already discussed in the context of RDFS and OWL in Section 2.4: rule-based inference and query rewriting. As we will see, the main problem in the context of attribute equations is that both of these approaches, when extended straightforwardly, would potentially not terminate. In the following, we present both of these implementation approaches and discuss their pros and cons.

7.2 Implementing Attribute Equations within Rules

Many rule-based approaches such as SWRL [43] offer additional support for mathematical built-ins.

Example 20

Using SWRL, (1) could be encoded as follows:

$$(?X, \text{dbo:revenueUSD}, ?USD) \leftarrow (?X, \text{dbo:revenueEUR}, ?EUR), ?USD = ?EUR * 1.3 \quad (3)$$

$$(?X, \text{dbo:profitEUR}, ?PEUR) \leftarrow (?X, \text{dbo:revenueEUR}, ?REUR), (?X, \text{dbo:totalExpensesEUR}, ?TEUR), ?PEUR = ?REUR - ?TEUR \quad (4)$$

◊

However, note that rules as exemplified above are not sufficient: (i) rule (3) is in the "wrong direction" for inferring additional results necessary for Query 1, that is, we would need different variants of the rule for converting from *EUR* to *USD* and vice versa; (ii) the above rules are not *DL safe* (i.e., we want to go beyond binding values only to explicitly named individuals where we also want to compute *new* values) which potentially leads to termination problems in rule-based approaches (and it actually does in existing systems).

Problem (i) could be solved in some cases by simply adding additional rules for each variant of each equation axiom, but the extended ruleset will, in turn, often give rise to Problem (ii), as shown.

Example 21

For the previous example, we can add more SWRL rules as follows:

$$(?X, \text{dbo:revenueEUR}, ?EUR) \leftarrow (?X, \text{dbo:revenueUSD}, ?USD), \quad ?EUR = ?USD/1.3 \tag{5}$$

$$(?X, \text{dbo:revenueEUR}, ?REUR) \leftarrow (?X, \text{dbo:profitEUR}, ?PEUR), (?X, \text{dbo:totalExpensesEUR}, ?TEUR), \quad ?REUR = ?PEUR + ?TEUR \tag{6}$$

$$(?X, \text{dbo:totalExpensesEUR}, ?TEUR) \leftarrow (?X, \text{dbo:revenueEUR}, ?REUR), (?X, \text{dbo:profitEUR}, ?PEUR), \quad ?TEUR = ?REUR - ?PEUR \tag{7}$$

However, here problem (ii) comes into play, as it is easy to see that these rules potentially produce infinite results, which we leave as an exercise to the reader. As a hint, consider the following example data:

```
:company1 dbo:profitEUR 1;
          dbo:revenueEUR 1;
          dbo:totalExpensesEUR 1;
```

Obviously, this data is not coherent with the equation, in the sense that it is ambiguous and a rule-engine that tries to compute the closure would not terminate (such example could occur in reality due to, e.g., rounding errors). ♢

Certain rule engines provide special built-ins to avoid running into non-termination problems as exemplified above. For instance, Jena provides a special built-in `noValue`, which returns sound but incomplete results whereby it only fires a rule if no value exists for a certain attribute on the inferences thus far or in the data – not unlike negation-as-failure.

Example 22

Using the `noValue` built-in, rule (4) (and analogously the other rule variants) could be encoded in Jena's rule syntax as follows:

```
[  (?X dbo:revenueEUR ?REUR) (?X dbo:totalExpensesEUR ?TEUR)
   difference(?REUR, ?TEUR, ?PEUR)  noValue(?X, dbo:profitEUR)
      -> (?X dbo:profitEUR ?PEUR)]
```

Values for `?PEUR` will only be computed from the given equations if no such value for `dbo:profitEUR` already exists on the resource bound to `?X`. ♢

7.3 Implementing Attribute Equations by Query Rewriting

An alternative implementation approach for reasoning with attribute equations (which according to initial experiments in [10] seems to work better than rules-based materialisation) is based on query rewriting – essentially extending Algorithm 1 from p. 16.

The idea here is that the expansion step in line 8 of Algorithm 1 is extended to also work with equation axioms as per Table 7. That is, informally, we extend the expansion function $\mathrm{gr}(g, i)$ from Table 2 as follows:

g	i	$\mathrm{gr}(g/i)$
(x, P_0, y)	$P_0 =; f(P_1, \ldots, P_n)$	$(x, P_1, ?V_{P_1}), \ldots, (x, P_n, ?V_{P_n}),\ A = f(?V_{P_1}, \ldots, ?V_{P_n})$

where we consider any equation axiom that has a P_0-variant. Similar to the rule-based approach, special care has to be taken such that equation axioms are not expanded infinitely; to this end, a simple blocking condition in the variant of Algorithm 1 presented in [10] avoids that the same equation axiom is used twice to compute the same value.

Example 23

To illustrate the approach, let us take a variation of Query 1 as an example, which only asks for the revenues of organisations, i.e., the SPARQL Query:

```
SELECT ?X ?R
WHERE { ?X a dbo:Organisation; dbo:revenueEUR ?R . }
```

We start with its formulation as a conjunctive query

$$q(?X, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Organisation}), (?X, \mathsf{dbo{:}revenueEUR}, ?R) \tag{8}$$

which is expanded as follows:

$$q(?X, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Organisation}), (?X, \mathsf{dbo{:}revenueEUR}, ?R) \tag{9}$$
$$q(?X, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Company}), (?X, \mathsf{dbo{:}revenueEUR}, ?R) \tag{10}$$
$$q(?X, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Organisation}), (?X, \mathsf{dbo{:}revenueUSD}, ?V_{revenueUSD}), \\ ?R = ?V_{revenueUSD}/1.3 \tag{11}$$
$$q(?X, ?R) \leftarrow (?X, \mathsf{a}, \mathsf{dbo{:}Company}), (?X, \mathsf{dbo{:}revenueUSD}, ?V_{revenueUSD}), \\ ?R = ?V_{revenueUSD}/1.3 \tag{12}$$

$\Diamond$

We note that translation back to SPARQL is not as straightforward here as it was without attribute equations, due to the fact that, as opposed to UCQs over only RDF triple predicates, we now are dealing with UCQs that also involve equality predicates and arithmetic operations such as $?R = ?V_{revenueUSD}/1.3$ in (11) and (12). Unions are again (like in Example 3) translated back to UNION

patterns in SPARQL, whereas equalities in query bodies are translated – depending on whether the left-hand side of these equalities is a variable or a constant – to either a BIND pattern[37], or a FILTER pattern.

Example 24

Following the previous example, this rewritten SPARQL 1.1 query will return the revenues of all three companies in our example data in EUR:

```
SELECT ?X ?R
WHERE { { ?X a dbo:Organisation; dbo:revenueEUR ?R .}
  UNION { ?X a dbo:Company; dbo:revenueEUR ?R .}
  UNION { ?X a dbo:Organisation; dbo:revenueUSD ?V_revenueUSD .
          BIND ( ?V_revenueUSD / 1.3 AS ?R ) }
  UNION { ?X a dbo:Company; dbo:revenueUSD ?V_revenueUSD .
          BIND ( ?V_revenueUSD / 1.3 AS ?R ) } }
```

◊

What we would like to emphasise here then, is that RDFS and OWL may not be enough for the reasoning requirements of Linked Data, where we show how (and why), e.g., numerical data can also be axiomatised for reasoning.

8 Summary

In this lecture we have illustrated particular challenges, opportunities and obstacles for applying OWL and RDFS reasoning in the context of querying Linked Data. We discussed the use of the RDFS and OWL standards in the area of Linked Data publishing, showing the degree to which individual features have been adopted on the Web. Though our results fall well short of indicating universal adoption, encouragingly, we find that many "lightweight" features of OWL and in particular RDFS have been widely adopted. We also provided practical examples as to how these RDFS and OWL axioms embedded in Linked Data can help for querying diverse sources, and how reasoning can thus help to further realise the vision of the Web of Data as one giant database.

However, while reasoning helps to obtain additional useful results in many cases, caution is required and specifically tailored reasoning algorithms need to be applied. In Sections 5–6 we have presented such tailored reasoning approaches and discussed their pros and cons; none of these approaches provides a panacea for reasoning "in the wild", the right approach depends a lot on the use case, particularly on the datasets considered and the query at hand. Still, the presented approaches have demonstrated that lightweight reasoning in di-

[37] BIND patterns are a new feature in SPARQL1.1 to assign values from an expression to a variable, see [30].

verse Linked Data setting is not only useful, but possible in practice, despite the enumerated challenges relating to scale, fallible data, inconsistencies, etc.

On the other hand, for modelling certain integration scenarios in Linked Data, we have shown that OWL and RDFS alone do not suffice to model a lot of implicit information and have briefly discussed attribute equations as an extension of OWL; given the increasing amount of published numerical data in RDF on the emerging Web of data, we believe that this topic deserves increased attention within the Semantic Web reasoning community. Generalising, though we have shown RDFS and OWL reasoning to be useful for querying Linked Data, these standards only provide a *basis* – not a solution – for capturing the semantics of Linked Data and still fall far short of that required to properly realise the vision of the Web of Data as a global, integrated database.

Acknowledgements

The work presented herein has been funded in part by Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2), by an IRCSET Scholarship. We thank our co-authors on joint works that flowed into this article, namely, Stefan Bischof, Piero Bonatti, Stefan Decker, Birte Glimm, Andreas Harth, Markus Krötzsch, Jeff Z. Pan, Luigi Sauro and Giovanni Tummarello.

References

1. D. Allemang and J. A. Hendler. *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. Morgan Kaufmann/Elsevier, 2008.
2. S. Auer, J. Lehmann, A.-C. N. Ngomo, and A. Zaveri. Introduction to Linked Data and its Lifecycle on the Web. In *Reasoning Web*. Springer, 2013. **(In this volume)**.
3. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application*. Cambridge University Press, 2002.
4. S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, W3C, February 2004.
5. D. Beckett and T. Berners-Lee. Turtle – Terse RDF Triple Language. W3C Team Submission, Jan. 2008. `http://www.w3.org/TeamSubmission/turtle/`.
6. D. Beckett, T. Berners-Lee, E. Prud'hommeaux, and G. Carothers. Turtle – Terse RDF Triple Language. W3C Candidate Recommendation, Feb. 2013. `http://www.w3.org/TR/2013/CR-turtle-20130219/`.
7. T. Berners-Lee. Linked Data. W3C Design Issues, July 2006. From `http://www.w3.org/DesignIssues/LinkedData.html`; retr. 2010/10/27.
8. T. Berners-Lee, R. T. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Jan. 2005. `http://tools.ietf.org/html/rfc3986`.
9. T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, 1999.
10. S. Bischof and A. Polleres. RDFS with attribute equations via SPARQL rewriting. In *Proceedings of the 10th ESWC*, LNCS. Springer (to appear), May 2013.

11. B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev, and R. Velkov. OWLIM: A family of scalable semantic repositories. *Semantic Web*, 2(1):33–42, 2011.
12. P. A. Bonatti, A. Hogan, A. Polleres, and L. Sauro. Robust and scalable Linked Data reasoning incorporating provenance and trust annotations. *J. Web Sem.*, 9(2):165–201, 2011.
13. D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-schema/`.
14. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated reasoning*, 39(3):385–429, 2007.
15. G. Cheng and Y. Qu. Integrating lightweight reasoning into class-based query refinement for object search. In *Asian Semantic Web Conference*, volume 5367 of *LNCS*, pages 449–463. Springer, 2008.
16. J. de Bruijn and S. Heymans. Logical foundations of (e)rdf(s): Complexity and reasoning. In *ISWC/ASWC*, volume 4825 of *LNCS*, pages 86–99. Springer, 2007.
17. R. Delbru, G. Tummarello, and A. Polleres. Context-dependent OWL reasoning in Sindice – experiences and lessons learnt. In *Web Reasoning and Rule Systems – Fifth International Conference, RR2011*, volume 6902 of *LNCS*, pages 46–60, Galway, Ireland, Aug. 2011. Springer.
18. I. Emmons, S. Collier, M. Garlapati, and M. Dean. RDF literal data types in practice. In *Proceedings of Workshop on Scalable Semantic Web Systems (SSWS)*, volume 5947 of *LNCS*. Springer, 2011.
19. R. T. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. `http://www.ietf.org/rfc/rfc2616.txt`.
20. F. Fischer, G. Ünel, B. Bishop, and D. Fensel. Towards a scalable, pragmatic knowledge representation language for the Web. In *Ershov Memorial Conf.*, pages 124–134, 2009.
21. B. Glimm. Using SPARQL with RDFS and OWL entailment. In *Reasoning Web*, volume 6848 of *LNCS*. Springer, 2011.
22. B. Glimm, A. Hogan, M. Krötzsch, and A. Polleres. OWL: Yet to arrive on the Web of Data? In *LDOW*. CEUR-WS.org (Vol. 937), 2012.
23. B. Glimm and C. Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation, Mar. 2013. available at `http://www.w3.org/TR/sparql11-entailment/`.
24. G. Gottlob and T. Schwentick. Rewriting ontological queries into small nonrecursive datalog programs. In *13th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR 2012)*, Rome, Italy, 2012. AAAI Press.
25. B. C. Grau, B. Motik, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language: Profiles. W3C Recommendation, Oct. 2009. `http://www.w3.org/TR/owl2-profiles/`.
26. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: combining logic programs with description logic. In *WWW*, pages 48–57, 2003.
27. R. V. Guha. *Contexts: a formalization and some applications*. PhD thesis, Stanford University, Stanford, CA, USA, 1992.
28. R. V. Guha, R. McCool, and R. Fikes. Contexts for the Semantic Web. In *International Semantic Web Conference*, volume 3298 of *LNCS*, pages 32–46. Springer, 2004.
29. H. Halpin, P. J. Hayes, J. P. McCusker, D. L. McGuinness, and H. S. Thompson. When owl:sameAs Isn't the Same: An Analysis of Identity in Linked Data. In

International Semantic Web Conference (1), volume 6496 of *LNCS*, pages 305–320. Springer, 2010.
30. S. Harris and A. Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, Mar. 2013. available at `http://www.w3.org/TR/sparql11-query/`.
31. A. Harth, S. Kinsella, and S. Decker. Using Naming Authority to Rank Data and Ontologies for Web Search. In *International Semantic Web Conference*, volume 5823 of *LNCS*, pages 277–292. Springer, 2009.
32. O. Hartig. SPARQL for a Web of Linked Data: Semantics and computability. In *ESWC*, volume 7295 of *LNCS*. Springer, 2012.
33. O. Hartig, C. Bizer, and J.-C. Freytag. Executing SPARQL queries over the Web of Linked Data. In *ISWC*, volume 5823 of *LNCS*. Springer, 2009.
34. O. Hartig and J.-C. Freytag. Foundations of traversal based query execution over linked data. In *HT*, pages 43–52. ACM, 2012.
35. P. Hayes. RDF Semantics. W3C Recommendation, Feb. 2004. http://www.w3.org/TR/rdf-mt/.
36. T. Heath and C. Bizer. *Linked Data: Evolving the Web into a Global Data Space (1st Edition)*, volume 1 of *Synthesis Lectures on the Semantic Web: Theory and Technology*. Morgan & Claypool, 2011. Available from `http://linkeddatabook.com/editions/1.0/`.
37. P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph. OWL 2 Web Ontology Language Primer. W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-primer/.
38. A. Hogan. *Exploiting RDFS and OWL for Integrating Heterogeneous, Large-Scale, Linked Data Corpora*. PhD thesis, Digital Enterprise Research Institute, National University of Ireland, Galway, 2011. Available from `http://aidanhogan.com/docs/thesis/`.
39. A. Hogan, A. Harth, and S. Decker. Performing Object Consolidation on the Semantic Web Data Graph. In *1st I3 Workshop: Identity, Identifiers, Identification Workshop*, volume 249 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
40. A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and browsing Linked Data with SWSE: The Semantic Web Search Engine. *J. Web Sem.*, 9(4):365–401, 2011.
41. A. Hogan, J. Z. Pan, A. Polleres, and S. Decker. SAOR: Template Rule Optimisations for Distributed Reasoning over 1 Billion Linked Data Triples. In *International Semantic Web Conference (ISWC)*, volume 6496 of *LNCS*, pages 337–353. Springer, 2010.
42. A. Hogan, A. Zimmermann, J. Umbrich, A. Polleres, and S. Decker. Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. *J. Web Sem.*, 10:76–110, 2012.
43. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3C member submission, W3C, 2004.
44. W. Hu, Y. Qu, and X. Sun. Bootstrapping object coreferencing on the Semantic Web. *J. Comput. Sci. Technol.*, 26(4):663–675, 2011.
45. T. Käfer, A. Abdelrahman, J. Umbrich, P. O'Byrne, and A. Hogan. Observing Linked Data dynamics. In *ESWC*. Springer (to appear), 2013.
46. R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to ontology-based data access. In *22nd Int'l Joint Conf. on Artificial Intelligence (IJCAI2011)*, pages 2656–2661, Barcelona, Catalonia, Spain, 2011. IJCAI/AAAI.

47. R. Kontchakov, M. Rodríguez-Muro, and M. Zakharyaschev. Ontology-based data access with databases: A short course. In *Reasoning Web*. Springer, 2013. **(In this volume)**.
48. A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On Blank Nodes. In *Proceedings of the 10th International Semantic Web Conference (ISWC 2011)*, volume 7031 of *LNCS*, Bonn, Germany, Oct. 2011. Springer.
49. F. Manola, E. Miller, and B. McBride. RDF Primer. W3C Recommendation, Feb. 2004. `http://www.w3.org/TR/rdf-primer/`.
50. J. Mayfield and T. Finin. Information retrieval on the Semantic Web: Integrating inference and retrieval. In *Proceedings of the SIGIR Workshop on the Semantic Web*, August 2003.
51. A. Miles, T. Baker, and R. Swick. Best Practice Recipes for Publishing RDF Vocabularies. W3C working group note, W3C, 2008.
52. B. Motik, P. F. Patel-Schneider, and B. C. Grau. OWL 2 Web Ontology Language Direct Semantics. W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-direct-semantics/.
53. S. Muñoz, J. Pérez, and C. Gutierrez. Simple and Efficient Minimal RDFS. *J. Web Sem.*, 7(3):220–234, 2009.
54. E. Oren, R. Delbru, M. Catasta, R. Cyganiak, H. Stenzhorn, and G. Tummarello. Sindice.com: a document-oriented lookup index for open linked data. *IJMSO*, 3(1):37–52, 2008.
55. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
56. P. F. Patel-Schneider, B. Motik, B. Cuenca Grau, I. Horrocks, B. Parsia, A. Ruttenberg, and M. Schneider. OWL 2 Web Ontology Language: Mapping to RDF Graphs. W3C Recommendation, Oct. 2009. `http://www.w3.org/TR/owl2-mapping-to-rdf/`.
57. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):Article 16 (45 pages), 2009.
58. H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.
59. A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *LNCS*, Budva, Montenegro, June 2006. Springer.
60. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. W3C Recommendation, Jan. 2008. `http://www.w3.org/TR/rdf-sparql-query/`.
61. R. Rosati. Prexto: Query rewriting under extensional constraints in DL-Lite. In *9th Extended Semantic Web Conf. (ESWC 2012)*, volume 7295 of *LNCS*, pages 360–374. Springer, 2012.
62. R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *12th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR2010)*. AAAI Press, 2010.
63. S. Rudolph. Foundations of Description Logics. In *Reasoning Web*, volume 6848 of *LNCS*. Springer, 2011.
64. M. Schneider. OWL 2 Web Ontology Language RDF-Based Semantics. W3C Recommendation, Oct. 2009. http://www.w3.org/TR/owl2-rdf-based-semantics/.
65. H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.

66. J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, 1989.
67. J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Improving the recall of live linked data querying through reasoning. In M. Krötzsch and U. Straccia, editors, *Web Reasoning and Rule Systems (RR)*, volume 7497 of *LNCS*, pages 188–204, Vienna, Austria, Sept. 2012. Springer.
68. J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. E. Bal. WebPIE: A Web-scale Parallel Inference Engine using MapReduce. *J. Web Sem.*, 10:59–75, 2012.
69. J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. Scalable distributed reasoning using MapReduce. In *International Semantic Web Conference*, volume 5823 of *LNCS*, pages 634–649. Springer, 2009.
70. R. Volz. *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe, Germany, 2004.
71. D. Vrandečíc, M. Krötzsch, S. Rudolph, and U. Lösch. Leveraging Non-Lexical Knowledge for the Linked Open Data Web. *Review of April Fool's day Transactions (RAFT)*, 5:18–27, 2010.
72. J. Weaver and J. A. Hendler. Parallel Materialization of the Finite RDFS Closure for Hundreds of Millions of Triples. In *International Semantic Web Conference (ISWC)*, volume 5823 of *LNCS*, pages 682–697. Springer, 2009.
73. E. Yardeni and E. Y. Shapiro. A Type System for Logic Programs. *J. Log. Program.*, 10(1/2/3&4):125–153, 1991.